Anforderungsgetriebene Qualitätsmodellierung und -auswertung in kompositen Web-Mashups

Dissertation

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

eingereicht von

Dipl.-Medieninf. Andreas Rümpel

Fakultät Informatik
Technische Universität Dresden

Gutachter

Prof. Dr.-Ing. Klaus Meißner · Technische Universität Dresden
Prof. Dr. rer. nat. Stefan Wagner · Universität Stuttgart

Fachreferent

Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill · Technische Universität Dresden

Datum der Einreichung: 30. August 2018
Datum der Verteidigung: 24. April 2019

Zusammenfassung

Komposite Web-Mashups stellen durch die intelligente Verknüpfung von *User-Interface-Services* und anderen Web-Ressourcen einen Mehrwert für Anwendungsszenarien in verschiedenen Situationen des privaten und geschäftlichen Lebens bereit. Obwohl die Verwendung solcher Mashups bereits viele Nutzerzielgruppen erreicht und Anwendungsdomänen erschlossen hat, ist die bedarfsgerechte Auswahl von Anwendungsbausteinen und deren intelligente Komposition immer noch eine große Herausforderung.

In dieser Arbeit werden deshalb Konzepte für die verbesserte Durchführung des Erstellungsprozesses und die Nutzung kompositer Web-Mashups entwickelt und vorgestellt. Kernidee ist dabei die *Modellierung und Auswertung anpassbarer Qualitätsanforderungen*. Erstmals wird es mit Hilfe eines speziell auf die Belange kompositer Web-Mashups zugeschnittenen *Modells für Qualitätseigenschaften* ermöglicht, solche Anforderungen passgenau für die Auswahl von Anwendungsbausteinen und die automatisierte Auswertung zu nutzen. Neben der Spezifikation von Bedingungen und Vergleichswerten für bestimmte Eigenschaften erlaubt es das ebenfalls hier vorgestellte *Metamodell für Qualitätsanforderungen*, die Rahmenbedingungen der Auswertung sowie zuzuordnende Aktionen festzulegen. Schwerpunkte der Arbeit sind außerdem der *Prozess der qualitätsbewussten Komposition*, die Nutzung der resultierenden Web-Mashups sowie die dazu gehörende *Referenzarchitektur*. Die betrachteten Anwendungsszenarien decken insbesondere die Teilprozesse der Auswahl von Kompositionsfragmenten, die Erweiterung »in Betrieb« befindlicher Anwendungen sowie die intelligente Adaption innerhalb der Laufzeitplattform ab. Neben Werkzeugen zur Modellierung und der automatisierten Auswertung von Qualitätsanforderungen setzt die Referenzarchitektur das kontextsensitive Monitoring von Qualitätseigenschaften um. Ein weiterer Fokus liegt auf der *unscharfen Spezifikation anpassbarer Qualitätsanforderungen*, da zur Zielgruppe insbesondere auch Menschen ohne Programmiererfahrung, jedoch mit guter Kenntnis in aktuellen Web-Technologien und in der jeweiligen fachlichen Domäne, zählen. Diese Ausrichtung wird sowohl bei der Modellierung mit *Fuzzy-Mengen* als auch in Form von Interaktionskonzepten berücksichtigt.

Anhand typischer Anwendungsfälle und unter Zuhilfenahme einer implementierten Infrastruktur und Anwendungslandschaft werden die vorgestellten Konzepte validiert und optimiert. Dabei fließen sowohl die Meinungen von Nutzern und Experten als auch die Betrachtungen zur Komplexität ressourcenintensiver Vorgänge sowie die Ergebnisse von Performance-Analysen ein. Durch das Verwenden der Modelle, Konzepte, Prozesse und Architekturen dieser Arbeit wird somit eine verbesserte Erstellung, Verfeinerung und Nutzung bedarfsgerechter, situativer Mashup-Anwendungen mit Hilfe anpassbarer Qualitätsanforderungen ermöglicht.

Inhaltsverzeichnis

1 Motivation und Zielstellung **1**
- 1.1 Problemdefinition . . . 2
- 1.2 Forschungsthesen . . . 4
  - 1.2.1 Einheitliche Modellierung für Mashups typischer Qualitätseigenschaften 4
  - 1.2.2 Automatisierte Auswertung anpassbarer Qualitätsanforderungen . . . 5
  - 1.2.3 Überwachung und Durchsetzung von Qualitätsanforderungen . . . 5
- 1.3 Forschungsziele . . . 6
  - 1.3.1 Eigenschafts- und Anforderungsmodellierung . . . 6
  - 1.3.2 Spezifikation und Auswertung von Qualitätsanforderungen . . . 7
  - 1.3.3 Plattformintegration und Anwendungsadaption . . . 7
- 1.4 Abgrenzung . . . 8
- 1.5 Aufbau der Arbeit . . . 9

2 Grundlagen der Entwicklung und Nutzung kompositer Web-Mashups **11**
- 2.1 Charakteristika und Einordnung des Anwendungstyps . . . 11
- 2.2 Entwicklungsmethoden und Anwendungsszenarien . . . 16
  - 2.2.1 Entwicklungsmethoden für Web-Mashups . . . 16
  - 2.2.2 Szenarien der Nutzung und Entwicklung kompositer Web-Mashups . . . 18
- 2.3 Rollen im Entwicklungsprozess von Web-Mashups . . . 21
  - 2.3.1 Zur Entwicklungszeit . . . 22
  - 2.3.2 Zur Laufzeit . . . 23
  - 2.3.3 Einordnung der Rollen und Zielgruppen in der Mashup-Plattform . . . 24
- 2.4 Qualitätseigenschaften und -anforderungen im Kontext von Web-Mashups . . . 25

3 Stand der Forschung und Technik **27**
- 3.1 Normen und Standards für Qualitätsmodelle bei Softwareprodukten . . . 27
- 3.2 Strukturierung und Erstellung von Qualitätsmodellen . . . 30
- 3.3 Anforderungsmodellierung mit Aufgaben und Fuzzy-Mengen . . . 32
- 3.4 Bewertungskriterien und Übersicht der Cluster für existierende Arbeiten . . . 34
  - 3.4.1 Übersicht der Kriterien und Skala der Bewertung . . . 34
  - 3.4.2 Modellierung und Eignung von Qualitätseigenschaften . . . 34
  - 3.4.3 Modellierung und Formalisierung von Qualitätsanforderungen . . . 35
  - 3.4.4 Integration in Werkzeuge und Entwicklungsprozesse . . . 35
  - 3.4.5 Automatisierbare Auswertung von Qualitätsanforderungen . . . 36
  - 3.4.6 Adaption und Verwertung in der Plattform . . . 36
  - 3.4.7 Clusterweiser Überblick repräsentativer Arbeiten . . . 36

3.5 Qualität in Web-Mashups . . . 37
3.5.1 Ermittlung der Wertebelegung von Qualitätseigenschaften auf Anwendungsebene durch Aggregation . . . 39
3.5.2 Empfehlungssysteme und EUD bei Mashups . . . 41
3.5.3 Fazit zum Stand der Forschung und Technik zur Mashup-Qualität . . . 43
3.6 Qualitätsanforderungen im Web-Engineering . . . 44
3.7 Qualitätseigenschaften und -anforderungen bei der Auswahl und Komposition von Web-Services . . . 46
3.8 Qualitätsanforderungen in kompositen Softwaresystemen . . . 48
3.9 Fazit zum Stand der Forschung und Technik . . . 50

4 Modellierung von Qualitätseigenschaften für Mashups 53
4.1 Modellüberblick und Abhängigkeiten . . . 54
4.2 Anforderungen an das Eigenschaftsmodell . . . 56
4.3 Metamodell zur Strukturierung von Qualitätseigenschaften . . . 58
4.3.1 Concerns . . . 59
4.3.2 Typen der Wertermittlung für Qualitätseigenschaften . . . 61
4.4 Unscharfe Eigenschaftswerte mit Fuzzy-Mengen . . . 62
4.4.1 Zugehörigkeitsfunktionen mit Punktketten . . . 63
4.4.2 Nutzerdefinierte Konfiguration von Termen . . . 64
4.5 Nutzung von Qualitätseigenschaften in der Mashup-Plattform . . . 65
4.5.1 Integration bereitgestellter Eigenschaften . . . 65
4.5.2 Integration gesammelter Eigenschaften . . . 67
4.5.3 Integration gemessener Eigenschaften . . . 68
4.5.4 Aggregation von Qualitätseigenschaften auf Anwendungsebene . . . 69
4.6 Referenzmodell der für Mashups typischen Qualitätseigenschaften . . . 72
4.6.1 Eigenschaften mit Concerns . . . 73
4.6.2 Eigenschaften ohne bestimmte Concerns . . . 75
4.7 Zusammenfassung und Bewertung des Modells für Qualitätseigenschaften . . . 77

5 Festlegen und Auswerten von Qualitätsanforderungen 79
5.1 Herausforderungen im Umgang mit Anforderungen . . . 79
5.2 Qualitätsanforderungen in der Mashup-Architektur . . . 80
5.3 Aufbau von Qualitätsanforderungen . . . 82
5.3.1 Metamodell für Qualitätsanforderungen in Web-Mashups . . . 83
5.3.2 Komposition und Wichtung von Bedingungen . . . 85
5.3.3 Auswahl geeigneter Operatoren . . . 86
5.3.4 Ereignisse zum Auslösen der Anforderungsauswertung . . . 88
5.3.5 Serialisierung von Anforderungsinstanzen . . . 89
5.4 Erzeugen und Bearbeiten von Qualitätsanforderungen . . . 91
5.4.1 Empfehlungsmodus . . . 92
5.4.2 Überwachungsmodus . . . 92
5.4.3 Modifikationspunkte kompositer Bedingungen . . . 93
5.4.4 Favorisierte Anforderungen und Qualitätsprofile . . . 93

5.5 Auswertung von Qualitätsanforderungen . . . 95
5.5.1 Rahmenbedingungen für die Auswertung . . . 96
5.5.2 Verteilter Auswertungsprozess . . . 96
5.5.3 Auswertungsalgorithmus . . . 97
5.6 Zusammenfassung . . . 99

6 Qualitätsbewusster Entwicklungs- und Nutzungsprozess 101
6.1 Entwicklung und Nutzung von Web-Mashups mit Qualitätsanforderungen . . . 102
6.1.1 Entwicklungsprozess für komposite Web-Mashups . . . 102
6.1.2 Verbesserung von Entwicklungsaufgaben und Ausführung durch Qualitätsanforderungen . . . 103
6.1.3 Qualitätsbewusste Entwicklung in den Anwendungsszenarien . . . 105
6.1.4 Diskussion von Entwicklungsparadigmen für Web-Mashups . . . 108
6.2 Aktionen im Kontext von Qualitätsanforderungen . . . 110
6.2.1 Entstehung und strukturelle Einordnung . . . 110
6.2.2 Referenzmodell möglicher Arten durchführbarer Aktionen . . . 110
6.2.3 Konflikte zwischen Adaptionsaktionen . . . 115
6.2.4 Lebenszyklus von Aktionen und Anforderungen . . . 117

7 Validierung und Implementierung 119
7.1 Validierungsmethodik . . . 119
7.2 Überblick der implementierten Infrastruktur . . . 121
7.3 Implementierung von Kompositionsfragmenten . . . 122
7.3.1 Angereicherte Mashup-Komponenten . . . 123
7.3.2 Angereicherte Mashup-Anwendungen . . . 124
7.4 Implementierung der Referenzarchitektur . . . 125
7.4.1 Fuzzy-Mengen und Fuzzy-Terme . . . 126
7.4.2 Evaluator für Qualitätsanforderungen . . . 126
7.4.3 Verwaltung von Qualitätseigenschaften . . . 127
7.4.4 Kontextsensorik . . . 128
7.4.5 Services in der Mashup-Plattform . . . 128
7.5 Implementierung der Werkzeuge . . . 130
7.5.1 Anforderungsassistent . . . 130
7.5.2 Editor für Fuzzy-Mengen . . . 132
7.5.3 Anwendungsbrowser mit Aggregationssicht und Komponentenbrowser 134
7.6 Nutzerstudie zum Anforderungsassistenten . . . 135
7.6.1 Szenarien und Methodik . . . 136
7.6.2 Ergebnisse und Konsequenzen für den Assistenten . . . 138
7.7 Validierungsergebnisse zu Performance und Awareness-Indikatoren . . . 139
7.8 Diskussion zur Validierung und Implementierung . . . 141

8 Zusammenfassung, Diskussion, Bewertung und Ausblick 143
8.1 Zusammenfassung der Kapitel . . . 143
8.2 Diskussion und Bewertung der Forschungsergebnisse . . . 147
8.2.1 Modellierungslandschaft . . . 148

8.2.2 Automatisierte Anforderungsauswertung 148
8.2.3 Spezifikation mit umgangssprachlichen Begriffen 148
8.2.4 Überwachung und Durchsetzung 149
8.2.5 Integrierter Entwicklungs- und Nutzungsprozess 149
8.3 Ausblick auf aktuelle und künftige Arbeiten 150

A Metamodelle und Schemata **153**

B Referenzmodelle **161**

C Komponentenimplementierungen **179**

D Werkzeuge **183**

E Dienste, Verwaltung und Tests **185**

Abbildungsverzeichnis

1.1 Ableitung von Herausforderungen und Kernproblemen bei der bedarfsgerechten Komposition und Nutzung kompositer Web-Mashups . . . 3
1.2 Dreiteilung der Kapitel dieser Arbeit in: Beschreibung der Ausgangssituation, Vorstellung der Konzepte und Reflexion der Ergebnisse . . . 10

2.1 Projektion von SOA auf Web-Mashups . . . 12
2.2 Zu Web-Mashups verwandte Architektur- und Anwendungstypen . . . 14
2.3 Überblick der Referenzinfrastruktur für komposite Web-Mashups . . . 15
2.4 Szenarien der Entwicklung und Nutzung von Web-Mashups mit Qualitätsanforderungen über Komponenten- und Kompositionsmodellen . . . 18

3.1 Typen von Qualitätsanforderungen nach [ISO25030] . . . 29
3.2 Strukturierung von Qualitätsmodellen mit FCM nach [Bal98] . . . 30
3.3 Metamodell zu ABQM nach [Loc10] . . . 31
3.4 User Interface in Ask Fuzzy [KKA12] – auf der rechten Seite oben die Verwendung unscharfer Terme (*low* und *high*), darunter die Visualisierung der Membershipfunktionen . . . 33
3.5 Bewertungsskala für existierende Ansätze . . . 34
3.6 Cluster betrachteter Arbeiten und Ansätze . . . 37
3.7 Qualitätsmodell für Mashup-Komponenten nach [CDM09] . . . 38
3.8 Qualitätsmodell für Mashups nach [Cap+11a] . . . 39
3.9 Kompositionsmuster für Mashups aus [Cap+10] . . . 40
3.10 Architektur des Empfehlungssystems in PEUDOM nach [Cap+12] . . . 42
3.11 2Q2U-Qualitätsframework aus [Ols+12] . . . 45
3.12 CQML$^+$-Ausschnitte aus [RZ07] . . . 49

4.1 Abhängigkeiten der semantischen Qualitätsmodelle und Referenzmodelle untereinander sowie zu externen Domänenmodellen . . . 55
4.2 Metamodell der Qualitätseigenschaften für komposite Web-Mashups . . . 58
4.3 Concerns der Qualitätseigenschaften bei Web-Mashups . . . 60
4.4 Wertermittlung von Qualitätseigenschaften in der Mashup-Infrastruktur . . . 61
4.5 Aktualisierung der Wertebelegung bei gemessenen Eigenschaften am Beispiel des Typs *ResponseTime* . . . 62
4.6 Schrittweise Konstruktion einer Zugehörigkeitsfunktion für Fuzzy-Mengen auf Basis einer Punktkette . . . 63
4.7 Konfiguration von Zugehörigkeitsfunktionen . . . 65
4.8 Integration bereitgestellter Werte für Qualitätseigenschaften im Komponentendeskriptor (Semantic Mashup Component Description Language (SMCDL)) . . . 66

4.9 Eigenschaftsaggregation auf Anwendungsebene mit ausgewählten Bestandteilen der Architektur für komposite Web-Mashups 70
4.10 Prozess der Aggregation von Qualitätseigenschaften 71

5.1 Einordnung der Infrastruktur zur Spezifikation und Auswertung von Qualitätsanforderungen sowie der Modelle in die Mashup-Architektur 81
5.2 Typische Kombinationen der Quellen und Ziele von Qualitätsanforderungen . . 82
5.3 Metamodell für Qualitätsanforderungen in kompositen Web-Mashups 84
5.4 Schematische Darstellung der Modi des Anforderungsassistenten 91
5.5 Iterative Komposition von Conditions über Erweiterungspunkte 94
5.6 Verwaltung und Nutzung favorisierter Qualitätsanforderungen in der verteilten Laufzeitplattform für Mashups unter Nutzung eines Kontextdienstes 94
5.7 Beispiele für Qualitätsprofile als vorgefertigte Anforderungen mit Label-Text und visueller Unterstützung 95
5.8 Auswertungsprozess für Qualitätsanforderungen mit mehreren Bedingungen (*c1* bis *c3*) auf verschiedenen Eigenschaftstypen und den Auswertungsergebnissen *r1* bis *r5* 98
5.9 Auswertungsprozess für Qualitätsanforderungen im Evaluator 98

6.1 Entwicklungs- und Nutzungsprozess für komposite Web-Mashups 102
6.2 Anknüpfungspunkte bei der Verbesserung von Entwicklungs- und Nutzungsprozessen kompositer Web-Mashups durch Qualitätsanforderungen 104
6.3 *Action* als Teil des ECA-Musters im Metamodell für Qualitätsanforderungen . . 111
6.4 Gruppen von Aktionen im Referenzmodell mit Kürzeln 115
6.5 Konfliktmatrix zwischen jeweils zwei zeitnah eintretenden Aktionen 116
6.6 Lebenszyklus von Aktionen in der Mashup-Plattform 118

7.1 Wesentliche Teile der implementierten Mashup-Infrastruktur: *Neuerungen* → durchgezogene Rahmen sowie *Erweiterungen* → gestrichelte Rahmen 121
7.2 Beispielübersicht von Komponenten im Repository 124
7.3 Komposite Nutzerschnittstelle einer Beispielanwendung aus der Domäne Tourismus in der CRUISE-Plattform 125
7.4 Serialisierung von Fuzzy-Sets im Datenmodell als TURTLE und der zugehörige Funktionsgraph 126
7.5 Anforderungsassistent im Empfehlungsmodus zum Spezifizieren von Fähigkeiten und Qualitätsanforderungen für neue Kompositionsfragmente 131
7.6 Anforderungsassistent im Überwachungsmodus zum Spezifizieren von Laufzeitanforderungen mit Überwachungsereignissen und Adaptionsaktionen 132
7.7 Fuzzy-Set-Editor: Filtern und Auswählen von Zugehörigkeitsfunktionen auf Basis von Eigenschaftstypen und Termen 133
7.8 Fuzzy-Set-Editor: Graphisches Bearbeiten einer Zugehörigkeitsfunktion 134
7.9 Anwendungsbrowser: Anwendungsübersicht 135
7.10 Anwendungsbrowser: Wertebelegungen der Qualitätseigenschaften für eine zuvor ausgewählte Mashup-Komponente 136
7.11 Anforderungsassistent in der Thin-Server-Laufzeitumgebung 137

7.12 Vergleich von Indikatoren für den Erfüllungsgrad von Bedingungen 140

D.1 Anwendungsbrowser: Detailansicht zur Aggregation verschiedener Werte der Qualitätseigenschaften 183
D.2 Condition-Editor mit Fuzzy-Bedingung und als Favorit markierter Anforderung 184
D.3 Aktivieren des Anforderungsassistenten im Überwachungsmodus für verschiedene Scopes: *Anwendung* (links über Plattform-Menü) und *Komponente* (rechts über Titelleisten-Icon) 184

E.1 Interaktives Browsen und Erproben der Bestandteile des REST-APIs im Component Repository (CoRe) mit Swagger-UI 187
E.2 Ausschnitt aus dem Swagger-UI des *REST*-basierten Web-Service zur Manipulation der Werte gesammelter Qualitätseigenschaften 188

Tabellenverzeichnis

3.1 Bewertung der Arbeiten zur Qualität in Web-Mashups 44
3.2 Bewertung der Arbeiten zu Qualitätsanforderungen im Web-Engineering 46
3.3 Bewertung der Arbeiten zur Komposition von Web-Services 48
3.4 Bewertung der Arbeiten zu Anforderungen in kompositen Softwaresystemen . . 50
3.5 Übersicht der Bewertungen ausgewählter Ansätze 51

4.1 Überblick der semantischen Qualitätsmodelle 54
4.2 Modifizierte und importierte semantische Modelle 55
4.3 Typische Funktionen für die Aggregation von Werten für Qualitätseigenschaften abhängig von deren Datentypen 71
4.4 Vorgabewerte für Qualitätseigenschaften nach Datentyp 72
4.5 Qualitätseigenschaften mit Zuordnung bestimmter Concerns 76

5.1 Referenzmodell für Ereignisse zum Auslösen der Auswertung von Bedingungen in Qualitätsanforderungen 89

Verzeichnis der Codeausschnitte

4.1 Deskriptor (SMCDL-Instanz) einer Mashup-Komponente für die Übersicht einer Beratungssitzung mit Wertebelegungen für Qualitätseigenschaften 66

5.1 Diskrete TURTLE-Qualitätsanforderung, gegen die Kandidaten von Kompositionsfragmenten geprüft werden können 89
5.2 Qualitätsanforderung mit mehreren Fuzzy-Termen 89
5.3 Qualitätsanforderung mit kompositer Bedingung 90
5.4 Qualitätsanforderung mit Concern, Event und Action 90

7.1 Ausschnitt aus dem Kontextmetamodell um Qualitätsanforderungen für Nutzer als Favorit zu speichern 129

A.1 Ausschnitt aus der XML Schema Definition (XSD) der erweiterten Metadaten für Komponentendeskriptoren in der SMCDL 153
A.2 Als TURTLE serialisiertes Metamodell für Qualitätseigenschaften mit dem Referenzmodell für Concerns 154
A.3 Als TURTLE serialisiertes Metamodell für Qualitätsanforderungen 156
A.4 Als TURTLE serialisiertes Modell zur Aggregation von Werten für Qualitätseigenschaften auf Anwendungsebene mit Zuordnung von Funktionen auf Datentypen 158

B.1 Ausschnitt aus dem Referenzmodell für Ereignisse zum Auslösen der Auswertung von Qualitätsanforderungen 174
B.2 Ausschnitt aus dem Referenzmodell für Aktionen zur Adaption in Abhängigkeit des Ausganges der Auswertung von Qualitätsanforderungen 174

C.1 Beispielimplementierung einer Mashup-Komponente für das Anzeigen des aktuellen Wetters an einem bestimmten Ort in JavaScript 179
C.2 Vorlage für die Implementierung von Mashup-Komponenten in JavaScript mit Fallbacks für den Lebenszyklus 181

E.1 Ausschnitt aus einem Unit-Test für das automatisierte Setup und Auswerten von Qualitätsanforderungen 185

Abkürzungsverzeichnis

ABQM	Activity-Based Quality Model
API	Application Programming Interface
AppRe	Application Repository
ASG	Abstract Syntax Graph
CBSE	Component-Based Software Engineering
CDN	Content Delivery Network
CoRe	Component Repository
CQML	Component Quality of Service Modeling Language
CRUISE	Composition of Rich User Interface Services for Everybody
CRUISe	Composition of Rich User Interface Services
CSS	Cascading Style Sheets
DC	Dublin Core
DIN	Deutsches Institut für Normung
DOM	Document Object Model
DRY	Don't Repeat Yourself
ECA	Event Condition Action
ECL	Energy Contract Language
EDA	Event-Driven Architecture
EDYRA	Engineering of Do-it-Yourself Rich Internet Applications
EN	Europäische Norm
EUD	End User Development
FCM	Factor-Criteria-Metrics
GQM	Goal Question Metric
GUI	Graphisches UI
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
JAX-RS	Java API for RESTful Web Services

JSON	JavaScript Object Notation
JSON-LD	JSON for Linking Data
JSR	Java Specification Request
MCM	Mashup Composition Model
MRE	Mashup Runtime Environment
NASA	National Aeronautics and Space Administration
NASA-TLX	NASA Task Load Index
NLP	Natural Language Processing
OCL	Object Constraint Language
OWL	Web Ontology Language
PaaS	Platform as a Service
POI	Point of Interest
QoS	Quality of Service
QUDT	Quantities, Units, Dimensions and Data Types Ontologies
RAM	Random Access Memory
RDF	Resource Description Framework
REST	Representational State Transfer
SaaS	Software as a Service
SI	Internationales Einheitensystem
SLA	Service Level Agreement
SLOC	Source Lines of Code
SMCDL	Semantic Mashup Component Description Language
SOA	Serviceorientierte Architektur
SoWa	Serviceorientierte Web-Anwendung
SPARQL	SPARQL Protocol and RDF Query Language
SUS	System Usability Scale
SWS	Semantic Web Services
TLS	Transport Layer Security
TURTLE	Terse RDF Triple Language
UI	User Interface
UML	Unified Modeling Language
URI	Universal Resource Identifier
USDL	Unified Service Description Language
VoD	Video on Demand
VOWL	Visual Notation for OWL Ontologies

W3C	World Wide Web Consortium
WFB	Weighted Faceted Browsing
WSDL	Web Services Description Language
WSMO	Web Service Modeling Ontology
XML	Extensible Markup Language
XSD	XML Schema Definition

Kapitel 1

Motivation und Zielstellung

Auf webbasierten Ressourcen und Diensten aufbauende Mashup-Anwendungen erfreuen sich einer zunehmenden Verbreitung und Beliebtheit im Verbraucher- und auch im Geschäftsbereich. Durch die Unterstützung komplexer Interaktionsformen webbasierter Technologien und moderner Browser können so in Kombination mit dem Mashup-Paradigma komposite *Rich Internet Applications* entstehen. Begünstigend hierfür ist die freie Verfügbarkeit vieler Application Programming Interfaces (APIs), vgl. *ProgrammableWeb* [Pro11], und die Nutzung dienstorientierter Paradigmen im Unternehmensumfeld [Pou09]. Je nach Fähigkeiten, Anwendungszielen und Komplexität ist es verschiedensten Personengruppen möglich, Mashups zu entwickeln und zu nutzen. Es handelt sich dabei nicht mehr nur um reine Daten-Mashups, wie etwa *Yahoo! Pipes* [Yah11], vielmehr nimmt die Integration von Anwendungsbestandteilen mit Benutzerschnittstelle eine zentralere Position ein. Besonders beliebt unter diesen *UI-Mashups* sind kartenbasierte Kompositionen, mit denen man sehr einfach Geoinformationsanwendungen bauen kann, vgl. beispielsweise *MashupAustralia* [Gov09, Unterabschnitt 7.4.1]. Neben der geschäftlichen Nutzung und der Integration unternehmensinterner Dienstleistungen gewinnen Mashups auch im Behörden- und Verwaltungsbereich, die sich aus Bausteinen unterschiedlicher Anwendungsschichten bedienen, an Bedeutung [Sch11].

Dem Wunsch der einfachen, situativen Integration einiger weniger Informationsquellen stehen zunehmende Anwendungskomplexität und Anforderungen aus Szenarien des Geschäftsumfeldes, wie etwa Skalierbarkeit, entgegen. Daher entwickelten sich bereits Ansätze, die einerseits die durch Endnutzer getriebene Anwendungserstellung mit intuitiven und zum Teil visuellen Techniken aus vorgefertigten Bestandteilen verfolgen, vgl. [Cap+11c]. Andererseits beschäftigen sich Lösungen für Enterprise-Mashups mit der Einbindung von Geschäftsprozessen und Möglichkeiten zur Modellierung von Anwendungslogik. Die zur Entwicklung von Web-Mashups notwendigen Kenntnisse und Fähigkeiten reichen somit je nach Einsatzdomäne, Plattform, erreichbarer Anwendungskomplexität und Entwicklungswerkzeugen von denen eines normalen Nutzers von Web-Anwendungen über die eines Mitarbeiters einer Fachabteilung in einem Unternehmen bis hin zu denen eines Entwicklers mit substanziellen Fähigkeiten und Erfahrungen in der Programmierung bzw. Modellierung.

Sowohl bei der nutzergetriebenen Erstellung von Anwendungen, dem End User Development (EUD), als auch bei der institutionellen Entwicklung und Nutzung ergibt sich ein gesteigertes Bedürfnis nach der Spezifikation von *Anforderungen an die Qualität der Mashup-Anwendungen*. Etablierte Techniken zur Anforderungsspezifikation und Modellierungsansätze für Qualitätseigenschaften sind dabei aufgrund des dienstorientierten Anwendungscharakters sowie des in Nutzungszeit und Entwicklungszeit verschmelzenden Entwicklungsparadigmas nicht ohne Weiteres nutzbar. Neben der Verantwortung, aus universell einsetzbaren Komponenten einen

Mehrwert durch die Anwendungskomposition zu erreichen, übernimmt der Entwickler auch die Verantwortung der qualitativen Sicherstellung bestimmter Eigenschaften, die im Rahmen der Anwendungsnutzung entstehen. Eine zeitaufwendige Zertifizierung durch eine dedizierte Testabteilung, wie sie in der klassischen Softwareentwicklung bei vorproduzierten Anwendungen praktiziert wird, ist daher nicht gegeben, da vor allem der *Long Tail of User Needs* für situative Anwendungsprobleme bedient wird. Im Kompositionsprozess von Mashups selbst entstehen außerdem für den Anwendungstyp spezifische Qualitätsanforderungen, die beispielsweise die Empfehlung passender Mashup-Komponenten als Kandidaten für die gewünschte Anwendung verfeinern. Ein Finanzinvestor baut sich seine Informationsanwendung, die ihn mit Marktdaten versorgt vermutlich eher aus Komponenten, die verlässliche Quellen anbinden, wobei ihm der Preis der Bestandteile egal ist. Im nächsten Schritt filtert er zusätzlich nach kurzen Antwortzeiten der angebundenen Daten-Services, sodass ein iterativer Empfehlungsprozess für nutzerspezifische Qualitätsanforderungen gefordert ist. Eine Zusicherung verschiedener Dienstleistungsqualitäten hat zudem im professionellen Einsatz kommerzielle Bedeutung, etwa beim Anbieten verschieden vergüteter Qualitätsprofile für den Kunden.

Es zeigt sich, dass Qualitätsanforderungen bei der Entwicklung und Nutzung von Mashups durch sehr unterschiedliche Motivationen entstehen, da die Ausgangssituationen der Nutzer in der Mashup-Plattform vielfältig sind. Als Gemeinsamkeit bleibt jedoch, dass sie als Einschränkungen bzw. *Bedingungen auf bestimmten Eigenschaften* betrachtet werden können. Dabei nehmen sie nicht nur auf funktionale und Qualitätseigenschaften der Mashup-Anwendung bzw. der Mashup-Komponenten – wie etwa die Zusicherung von Dienstantwortzeiten oder Nutzerbewertungen – Bezug, sondern auch auf Eigenschaften des Nutzerkontextes sowie der Laufzeitplattform. Dazu gehören auch die Parameter beteiligter Geräte, beispielsweise die Displaygröße oder beliebte Anwendungsmuster eines Nutzers. Ein Beispiel für die Nutzung einer anforderungsbewussten Mashup-Plattform abseits der iterativen Erstellung nutzerdefinierter Anwendungen ist das *Testen von Plausibilitätskriterien* oder das *Überwachen von Zustandsgrößen zur Laufzeit*, wie etwa das Monitoring eines als Testdatensatz festgelegten Temperaturbereiches bei einer Wetterkomponente, die die aktuelle Lufttemperatur anzeigt.

Modellbasierte Ansätze zur Entwicklung von Web-Mashups unterstützen bisher allenfalls die Entgegennahme und Umsetzung von Anforderungen bezüglich offensichtlicher Metadaten, die durch die Definition ihrer Komponentenschnittstellen entstehen. Auf abstrakter Ebene sind diese Metadaten – beispielsweise ein Titel, eine Textbeschreibung oder eine Reihe von Keywords – auf aufgabenbasierte Annotationen des Nutzungszwecks der gewünschten Anwendung abbildbar, vgl. [Tie+11]. Für Anforderungen, die sich nicht unmittelbar auf den Zweck oder die Schnittstelle von Anwendungsbestandteilen beziehen, sondern als *Qualitätsanforderungen* darauf Bedingungen definieren, existiert bisher keine zufriedenstellende Lösung in Forschungsansätzen und Produkten zu kompositen Web-Mashups.

1.1 Problemdefinition

Obwohl technische Voraussetzungen wie die weite Verbreitung und die Kommunikationsanbindung von Geräten zur Ausführung kompositer Web-Mashups längst gegeben sind, mangelt es an *geeigneten Formen der bedarfsgerechten Spezifikation anpassbarer Qualitätsanforderungen* in aktuellen Mashup-Plattformen. Zusammen mit der Bestrebung, Kompositionsaufgaben und

zugeschnittene Nutzung dieser Anwendungen einem breiteren Personenkreis zu ermöglichen, stellt dies eine Grundherausforderung dar, die in dieser Arbeit betrachtet wird. Abbildung 1.1 verdeutlicht die Ableitung grundlegender Problemstellungen, die sich aus den genannten Herausforderungen der Anwendungserstellung und der intelligenten Ausführung kompositer Web-Mashups ergeben.

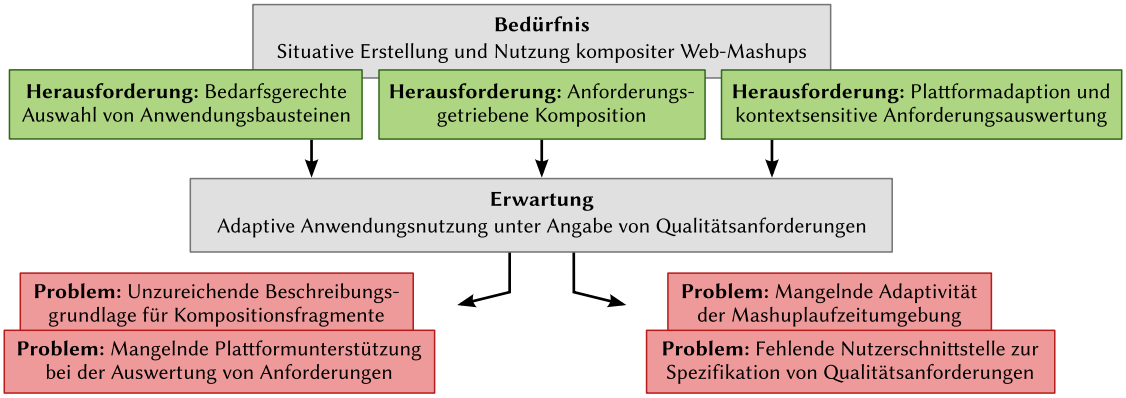

Abbildung 1.1: Ableitung von Herausforderungen und Kernproblemen bei der bedarfsgerechten Komposition und Nutzung kompositer Web-Mashups

Aus der Nutzererwartung der adaptiven Nutzung kompositer Anwendungen unter Angabe von Qualitätsanforderungen ergeben sich unter Verwendung bisheriger Ansätze *erwartungsnahe Kernprobleme*, vgl. rechter unterer Teil in Abbildung 1.1. Interaktionskonzepte zur nutzergetriebenen Spezifikation von Qualitätsanforderungen sind zwar aus anderen Bereichen des Web-Engineerings bekannt, jedoch nicht auf die Granularität und Belange der hier betrachteten Kompositionsfragmente abgestimmt. Annahme hierbei ist, dass in der berücksichtigten Art von Mashup-Plattform *Black-Boxes* als Anwendungsfragmente komponiert werden. Zum Erwartungsraum gehören auch einfachste Möglichkeiten des Browsens und Filterns solcher Anwendungsbausteine auf der Basis von Qualitätseigenschaften. Durch das bisher fehlende Bewusstsein aktueller Plattformen bzw. ihrer Nutzungsparadigmen für Qualitätsanforderungen lassen sie eine automatisierte Anforderungsauswertung vermissen. Folglich wird die intelligente, adaptive Anwendungsnutzung erschwert.

Ein dem Nutzer zunächst verborgenes, jedoch aus technischer Sicht essenzielles Problem ist die unzureichende Modellierungs- und Beschreibungsgrundlage auf verschiedenen Ebenen, vgl. linker unterer Teil in Abbildung 1.1. Komponentenmodelle, die komponierbare Anwendungsbestandteile in Aufbau und Fähigkeiten beschreiben, enthalten nicht die notwendigen Angaben, um sinnvolle Qualitätsanforderungen im Kompositionsprozess zu spezifizieren. Die erforderlichen Daten sollten Qualitätseigenschaften in maschinell zu erfassender und zu verarbeitender Form beinhalten. Situative Anwendungsszenarien fordern darüber hinaus auch ein hohes Maß an Automatisierung innerhalb einer Mashup-Plattform, da Anforderungen in vielen Fällen nur zwischen einer Person und der Mashup-Plattform ausgetauscht werden sollen. Somit repräsentiert die *geeignete Modellierung von Qualitätsanforderungen* ein Kernproblem, da umfassende Anforderungskonzepte in der Welt der Mashup-Modellierung Neuland sind. Es fehlt insbesondere ein Metamodell zur grundlegenden Durchdringung des Themas und Beschreibung

von Abhängigkeiten. Als aufbauende Herausforderung soll die erwünschte Modellierungs- und Auswertungsinfrastruktur auch für Überwachungstätigkeiten zur Laufzeit ohne den nutzerzentrierten Auswertungszyklus geeignet sein.

Aus der in den beschriebenen Problemen nicht erfüllbaren Erwartungshaltung der Entwickler und Nutzer kompositer Web-Mashups ergibt sich die Notwendigkeit für Forschungsanstrengungen, die nun anschließend durch Forschungsthesen, Ziele und zu erreichende Ergebnisse beschrieben werden. Im Vordergrund steht dabei, dass das initiale Bedürfnis der situativen Entwicklung und Nutzung kompositer Web-Mashups in verschiedenen Anwendungsfällen vereinfacht werden soll. Weitere, aus der beschriebenen Motivation hervorgehende Detailprobleme, die in dieser Arbeit nicht zentral verfolgt werden, sind in Abschnitt 1.4 abgegrenzt.

1.2 Forschungsthesen

Die beschriebene Ausgangssituation zeigt sowohl die zunehmende Akzeptanz des Paradigmas *Web-Mashup* zur Anwendungsnutzung und -entwicklung als auch die damit verbundenen Risiken und die daraus entstehenden Bedürfnisse zur Formulierung, Überwachung und kontrollierten Durchsetzung von Qualitätsanforderungen. Aus den genannten Herausforderungen und Problemen ergeben sich Forschungsthesen, die den inhaltlichen Kern der hier beschriebenen Forschungsarbeit umspannen. In Zuordnung zu den in diesem Abschnitt beschriebenen Thesen wird jeweils der angestrebte Lösungsansatz skizziert, der Grundlage für die Erreichung und Validierung der damit verbundenen Ziele ist.

1.2.1 Einheitliche Modellierung für Mashups typischer Qualitätseigenschaften

These 1: *Das Identifizieren und einheitliche Modellieren von für Web-Mashups typischen Qualitätseigenschaften **vereinfacht** den Zugriff auf qualitätsrelevante Basisdaten, die über unterschiedliche Quellen ermittelt werden, sowie deren anforderungsgetriebene Nutzung.*

In bekannten Qualitätsmodellen für Mashups und in Standards für Softwareprodukte werden Qualitätseigenschaften als sehr grob kategorisierte Optimierungsgrößen betrachtet. Angestrebt wird jedoch das Betrachten von Größen bzw. Eigenschaften, deren Qualitätsziele abhängig von Rollen, Anwendungsdomänen und Komponenten der Infrastruktur im Entwicklungs- und Nutzungsprozess durch Anforderungen ausgedrückt werden können. Existierenden Modellen für Web-Services fehlt zudem die Unterstützung UI-spezifischer Eigenschaften. Differenziert werden können solche Eigenschaften bzw. die zugrundeliegenden Basisdaten vor allem durch ihre Herkunft, einerseits statisch an Mashup-Komponenten und -Anwendungen gebunden, andererseits zur Laufzeit durch die Plattform ermittelt bzw. gemessen oder statistisch gesammelt. Unter Zuhilfenahme semantischer Modellierung können die heterogenen Bestandteile der Mashup-Infrastruktur als Grundlage für die übergreifende Formulierung von Qualitätsanforderungen dienen. Gefordert ist ein Modell für die einheitliche Repräsentation von Qualitätseigenschaften, das sowohl die für den Anwendungstyp charakteristischen Basisdaten berücksichtigt, als auch Metriken enthält, die Skalierung, Wertebereiche und Semantik beschreiben.

1.2.2 Automatisierte Auswertung anpassbarer Qualitätsanforderungen

These 2: *Die automatisierte Auswertung von Qualitätsanforderungen in einer Mashup-Plattform **erhöht** die vom Nutzer angestrebte Adaptivität kompositer Anwendungen zur Laufzeit und **unterstützt** ihre iterative Entwicklung.*

Die Auswertung beschriebener Qualitätsanforderungen kann sowohl zu unterschiedlichen Zeiten als auch in verschiedenen Infrastrukturkomponenten – wie etwa der Laufzeitumgebung oder einem Komponentenmarktplatz – erfolgen. Als Konsequenz der Auswertung werden korrespondierende Adaptionsaktionen eingeleitet, die – zur Situation passend – der Anforderung zugeordnet werden müssen. Eine Softwarekomponente zur Anforderungsauswertung hat dabei auch die Aufgabe, benötigte Kontextsensoren intelligent anzusprechen und zu steuern. Gefordert ist eine Möglichkeit zur maschinellen Auswertung von Qualitätsanforderungen, die dem System sowohl während der Entwicklung als auch zur Laufzeit bekannt werden. Entwicklungsarbeiten erfahren durch sie eine Unterstützung, indem die automatisierte Auswertung die Basis für Empfehlungsalgorithmen liefert.

These 3: *Die unscharfe Formulierung mit umgangssprachlichen Begriffen **erleichtert** Anwendungsnutzern ohne Programmiererfahrung die Spezifikation ihrer Qualitätsanforderungen.*

Es gilt eine für die Anforderungsquelle – beispielsweise den Anwendungsnutzer, den Entwickler oder den Domänenexperten – zugängliche Form der Spezifikation von Qualitätsanforderungen mit angemessener Komplexität zu finden. Eine besondere Herausforderung stellen dabei Nutzer ohne Programmier- oder Modellierungserfahrung dar, die Qualitätsanforderungen im Rahmen des EUD haben. Gefordert ist eine für die maschinelle Verarbeitung geeignete Form der Anforderungsspezifikation, die zugleich den Erstellungsprozess durch unerfahrene Web-Nutzer – etwa über einen Assistenten – unterstützt. Geeignete Interaktionstechniken im UI der Plattform sollen dem beschriebenen Fertigkeitsniveau entsprechen.

1.2.3 Überwachung und Durchsetzung von Qualitätsanforderungen

These 4: *Die Kopplung von Handlungsvorschriften an Bedingungen **erhöht** die Wirksamkeit von Qualitätsanforderungen.*

Modellierung und Nutzung von Handlungsvorschriften sind essenziell für die sinnvolle Verwendung von Qualitätsanforderungen. Dabei wird festgelegt, welche Konsequenzen die Nichteinhaltung von Anforderungen bezüglich ihrer enthaltenen Bedingungen hat. Die sich aus den Konsequenzen ergebenden Aktionen werden möglichst automatisch durchgeführt bzw. es werden entsprechende Schnittstellen angeboten. In vielen Fällen kann durch Adaption der Anwendung oder ihrer Komponenten angemessen reagiert werden. Allerdings ist auch die einfache Benachrichtigung des Nutzers in bestimmten Fällen eine sinnvolle Reaktionsmöglichkeit. Gefordert ist ein Referenzmodell möglicher Aktionen, ein Konzept zur Kopplung dieser an die Bedingungen der Qualitätsanforderungen und zur situationsgerechten Ausführung.

These 5: *Das Berücksichtigen von Nutzer- und Gerätekontext in Qualitätsanforderungen **vereinfacht** die nahtlose Entwicklung und Nutzung kompositer Anwendungen in einer Mashup-Plattform und **erhöht** somit die Effizienz im Entwicklungsprozess.*

Ein Hauptzweck von Qualitätsanforderungen ist neben der Steuerung der Anwendungsadaption vor allem die Empfehlung bei der Suche und Auswahl zu integrierenden Kompositionsfragmente im Erstellungsprozess von Mashups. Dabei sollen favorisierte Anforderungen sitzungsübergreifend angeboten werden. In mobilen Nutzungsszenarien gilt es außerdem, ortsabhängige und sonstige gerätespezifische Kontextinformationen zu berücksichtigen. Das auf schnelle Entwicklung und sofortige Nutzung ausgerichtete Mashup-Paradigma wird auf diese Weise unterstützt. Gefordert ist ein Konzept zum Einbeziehen von Nutzerpräferenzen und -profilen sowie weiterer Kontextparameter genutzter Geräte und Systeme im Rahmen der Ausführung kompositer Web-Mashups. Insbesondere wird die Integration in die Prozesse der automatisierten Auswertung und Spezifikation von Qualitätsanforderungen angestrebt.

1.3 Forschungsziele

Abgeleitet aus den im Fokus der Arbeit stehenden Problemen aus Abschnitt 1.1 werden nun Ziele vorgestellt, die mittels konzeptioneller und praktischer Forschungsergebnisse erreicht werden sollen. In ihrer Gesamtheit stellen die Forschungsziele eine Infrastruktur zur anforderungsgetriebenen Qualitätsmodellierung und automatisierten Auswertung der Qualitätsanforderungen für komposite Web-Mashups bereit. Die zu erreichenden Ziele ordnen sich in die Schwerpunkte *Modellierung, Spezifikation und Auswertung* sowie *Integration und Adaption* ein.

1.3.1 Eigenschafts- und Anforderungsmodellierung

Um die Ansprüche der in dieser Arbeit vorgeschlagenen Infrastruktur der Spezifikation und automatisierten Auswertung von Qualitätsanforderungen erfüllen zu können, bedarf es einer Beschreibungsform dieser Anforderungen, die gleichermaßen sowohl in Anwendungsfällen mit nutzerdefinierten Wünschen als auch für die automatisierte, maschinelle Verarbeitung geeignet ist. Basis zur Formulierung von Bedingungen innerhalb solcher Anforderungen ist ein Modell für typische Qualitätseigenschaften kompositer Web-Mashups.

Modell für Qualitätseigenschaften: Um eine bestmöglich zum Anwendungstyp passende Auswahl an Qualitätseigenschaften bereitzustellen, wird als erstes Ziel analysiert, welche dieser Eigenschaften aus verwandten Paradigmen und Plattformen in der Entwicklung kompositer Web-Mashups besonders relevant sind. Zusammen mit neu hinzugefügten Größen werden die Qualitätseigenschaften in einem Modell spezifiziert, das die formale Grundlage zur Nutzung innerhalb von Anforderungen bildet. Zudem ist es Ziel dieser Arbeit, die Modellelemente in das Komponenten- bzw. Kompositionsmodell einer bestehenden Mashup-Plattform zu integrieren und dadurch Synergien im Entwicklungs- und Nutzungsprozess zu schaffen.

Modell für anpassbare Qualitätsanforderungen: Hier gilt es, basierend auf der formalen Grundlage der um Qualitätseigenschaften angereicherten Modelle für Mashup-Komponenten und -Anwendungen, Anforderungen zu bilden. Dafür soll ein Metamodell für

Qualitätsanforderung entstehen, das sowohl beschreibt, welche Bedingung zur Entwicklungszeit oder zur Laufzeit zu überprüfen sind, als auch welche Aktionen damit verknüpft werden sollen und zu welcher zeitlichen oder auf andere Art und Weise definierten Gelegenheit die jeweilige Anforderung ausgewertet wird. Insbesondere soll innerhalb des Metamodells berücksichtigt werden, dass auch die Anwendungsfälle der unscharfen Spezifikation sowie der Komposition von Teilanforderungen über verschiedene Eigenschaften abgedeckt sind.

1.3.2 Spezifikation und Auswertung von Qualitätsanforderungen

Die Modellierungsgrundlage ermöglicht zunächst eine konzeptionelle Nutzung von Anforderungen. Für praktische Anwendungsfälle unter Verwendung einer Mashup-Plattform werden allerdings unbedingt noch Schnittstellen zur Spezifikation der Anforderungen – insbesondere auch in einer für Web-Nutzer geeigneten Art und Weise – und eine Softwarekomponente zur automatisierten Anforderungsauswertung benötigt.

Schnittstellen zum Erstellen von Qualitätsanforderungen: Als grundlegendes Ziel sollen in der Laufzeitplattform für Web-Mashups Schnittstellen geschaffen werden, die Qualitätsanforderungen in der dem Metamodell entsprechenden Form entgegennehmen können. Darüber hinaus ist es insbesondere im Rahmen der integrierten Nutzung und Verbesserung von Anwendungen, die in dieser Arbeit als *Live-Sophistication* etabliert wird, von großer Bedeutung, Benutzerschnittstellen zur Erstellung von Qualitätsanforderungen anzubieten. Als aufbauendes Ziel soll daher ein Werkzeug entstehen, das über für Web-Nutzer geeignete Interaktionsformen – wie das Browsen verfügbarer Eigenschaften und die visuelle Komposition – die Spezifikation von Anforderungen als Instanzen des Metamodells unterstützt. Diese Benutzerschnittstelle soll als Erweiterung der Mashup Runtime Environment (MRE) umgesetzt werden.

Automatisierte Anforderungsauswertung: Mit der formalen Spezifikation wird eine automatisierte Verarbeitung von Qualitätsanforderungen ermöglicht. Abhängig vom Anwendungsszenario muss die Anforderungsauswertung an bestimmten Stellen in der Infrastruktur, zu bestimmten auslösenden Gelegenheiten und zu einem bestimmten Zweck geschehen. Gelegenheiten ergeben sich beispielsweise aus Entwicklungsschritten in der Anwendung oder aus festgelegten Zeitspannen. Zur Umsetzung dieser Herausforderungen wird ein Evaluator angestrebt, der Instanzen des Metamodells für anpassbare Qualitätsanforderungen entgegennimmt, die Anforderung gegen tatsächliche Eigenschaftswerte verfügbarer Kompositionsfragmente auswertet und für das Einleiten zugeordneter Aktionen sorgt. Konzeptionelles Kernziel ist hierbei das Modellieren einer Struktur für Auswertungsvorschriften, die beispielsweise während des Empfehlungsprozesses notwendige Daten über die Eignung von Kompositionsfragmenten liefern.

1.3.3 Plattformintegration und Anwendungsadaption

Sowohl während der Live-Sophistication als auch bei der Ausführung existierender Mashups soll durch die Nutzung von Qualitätsanforderungen ein intelligenter Anwendungsbetrieb durch

Adaption des Mashups erfolgen. Diesem Gesamtziel wird durch die Zuordnung von Aktionen zu Bedingungen innerhalb von Anforderungen begegnet.

Kopplung von Aktionen an Qualitätsanforderungen: Ziel ist es zum einen, während der Ausführung des Mashups geeignete Maßnahmen der durch die Anforderung definierten Bedingungen zu ergreifen. Dieses Vorgehen ist mit Vertragsstrafen vergleichbar, die zuvor in einem Regelwerk festgelegt wurden. Zum anderen wird Plattformadaptivität – etwa durch Vorschlagsmechanismen bei Teilprozessen der Anwendungsentwicklung – angestrebt. In dieser Arbeit soll daher ein Referenzmodell und die Umsetzung wichtiger Aktionen vorgestellt werden, um als Ergänzung zu Qualitätsanforderungen in verschiedenen typischen Anwendungsfällen und -domänen sinnvolle Adaptionsmöglichkeiten anbieten zu können. Das Spektrum der Aktionen erstreckt sich dabei von automatischem Komponentenaustausch über Feedback an den Nutzer bzw. Entwickler bis hin zu komplexen Empfehlungsmechanismen, die alternative Bestandteile für das in der Entwicklung oder Ausführung befindliche Mashup vorschlagen können.

Integration von Kontextsensorik: Bei der Nutzung und Entwicklung sollen Kontextsensoren als Bestandteil der Laufzeitumgebung messbare Werte für Qualitätseigenschaften liefern, die in der Auswertung von Anforderungen als Vergleichswerte herangezogen werden können. Dadurch wird der Spielraum der Adaptionsfähigkeiten der Plattform und damit der kompositen Mashup-Anwendungen vergrößert. Die Kontextinformationen sollen dabei auch nutzer- und gerätespezifische Daten enthalten.

1.4 Abgrenzung

Das Forschungsthema der anforderungsgetriebenen Qualitätsmodellierung und -auswertung in kompositen Web-Mashups zeichnet sich durch eine starke Einbindung verwandter Forschungsfelder aus. Sowohl die Welt der Web-Anwendungen und Web-Services als auch die der Komposition von Benutzerschnittstellen sowie modellbasierte Entwicklungsprozesse für Web-Anwendungen sind mit dem hier vorgestellten Forschungsthema verzahnt. Zur Verdeutlichung des Nutzens der Ergebnisse dieser Arbeit für angrenzende Forschungsfelder werden nun die Schnittstellen und zugleich die Abgrenzungspunkte beschrieben.

Entwicklung von Mashup-Komponenten: Wie beim Entwickeln anderer kompositer Anwendungssysteme existiert auch beim Mashup-Paradigma eine Aufteilung der Entwicklungsaufwände zwischen der Modellierung und Umsetzung von Anwendungsbausteinen und der Anwendungsentwicklung aus diesen Komponenten. Ein wesentlicher Vorteil dieses Ansatzes ist die Möglichkeit der Verteilung von Entwicklungsaufgaben auf verschiedene Entwicklerrollen. Diese Arbeit widmet sich primär der intelligenten Komposition von Black-Box-Bestandteilen. Im Zuge der Optimierung dieser Kompositionstätigkeit – auch durch Methoden des EUD – werden Techniken eingesetzt, die möglicherweise den *Aufwand für die Entwicklung der Mashup-Komponenten* erhöhen. Deren Existenz, Verwendbarkeit und Beschreibung werden jedoch im Rahmen dieser Arbeit – sofern nicht abweichend angegeben – als gegeben angenommen, sodass Betrachtungen über den Aufwand für Komponentenentwickler hier von untergeordneter Bedeutung sind. Im Rollenmodell – siehe Abschnitt 2.3 – wird nochmals darauf Bezug genommen.

Primitive Überprüfungen durch Laufzeitumgebungen: Bestimmte Konzepte der Modellierung, die in den Grundlagen in Kapitel 2 näher vorgestellt werden, liefern bezüglich des *Sicherstellens der Übereinstimmung von Datentypen der Komponentenschnittstellen* in Zusammenarbeit mit der Laufzeitumgebung bereits primitive Lösungen. Diese Arbeit baut darauf auf und bietet darüber hinaus Qualitätsanforderungen auf solchen Bestandteilen des Interfaces an, die beispielsweise Bereichsprüfungen im Sinne der Plausibilität der Datenkommunikation innerhalb einer kompositen Anwendung ermöglichen.

Dekomposition von Aufgabenbäumen: Die Arbeit von Tietz u. a., vgl. [Tie+11], stellt eine Grundlage für die initiale Zerlegung von Nutzerwünschen und Intentionen dar, die eine situative Nutzung einer Mashup-Anwendung bietet. Das hier vorgestellte Konzept der intelligenten Komposition von Mashup-Anwendungen kann in *Ergänzung zur Dekomposition von Aufgaben* und deren folglicher Zuordnung betrachtet werden. In Kapitel 6 wird beim Vorstellen des verbesserten Entwicklungsprozesses derart darauf eingegangen, dass der Import von Fremdmodellen, die auch als Aufgabenmodelle existieren können, als zusätzlicher Einstieg in die anforderungsbasierte Entwicklung unterstützt wird.

Empfehlungssysteme als Anwendungsfall: Ein wichtiger Anwendungsfall für das in dieser Arbeit vorgestellte Konzept ist das Auffinden und Bewerten zu anpassbaren Qualitätsanforderungen passender Mashup-Komponenten. Die Arbeit liefert in diesem Zusammenhang eine *Erweiterung zum Empfehlungssystem der Mashup-Plattform* CRUISE, vgl. [Rad+12]. Dabei geht die hier vorgestellte Modellierungsgrundlage und Auswertungsinfrastruktur allerdings über diesen Anwendungsfall hinaus und ist universell für viele andere Szenarien, vgl. Aktionen in Kapitel 6, einsetzbar.

Mediation: Die im Grundlagenkapitel beschriebene semantische Schnittstellenbeschreibung stellt die Basis für eine *Datentypmediation* dar, die die Zusammenarbeit zunächst nicht exakt passender Anwendungsbestandteile innerhalb eines Web-Mashups ermöglicht, vgl. [Rad+14]. Diese Mediation und das hier vorgestellte Konzept zur anforderungsgetriebenen Auffindung von Mashup-Komponenten vor allem innerhalb der Live-Sophistication sind in Ergänzung zueinander zu sehen.

Prozessqualität: Diese Arbeit beschäftigt sich nicht mit der Qualität vorhandener Erstellungsprozesse, sondern mit der *Produktqualität* kompositer Anwendungen und ihrer Bestandteile sowie den Prozessen der Optimierung und Nutzung dieser. Eine detaillierte Abgrenzung wird in Abschnitt 2.4 vorgenommen.

1.5 Aufbau der Arbeit

Die insgesamt dreigeteilte Struktur dieser Arbeit, siehe Abbildung 1.2, erlaubt die folgende aufeinander aufbauende Betrachtung des Forschungsthemas der anforderungsgetriebenen Qualitätsmodellierung und -auswertung in der Entwicklung kompositer Web-Mashups. Die *Ausgangssituation* wird zusammen mit der Motivation und den Zielstellungen unter Berücksichtigung wichtiger Basistechnologien und -paradigmen sowie der Einordnung in existierende Arbeiten beschrieben. Anschließend stellt diese Arbeit die *konzeptionellen* und praktischen Ideen

bzw. Ergebnisse zur Modellierung, zur Infrastruktur und zu den Einsatzszenarien vor. Diese werden schließlich in Bezugnahme auf die Fortschrittlichkeit des beschriebenen Ansatzes validiert, zusammengefasst, diskutiert und bewertet. Im Einzelnen gestalten sich die Kapitel wie folgt:

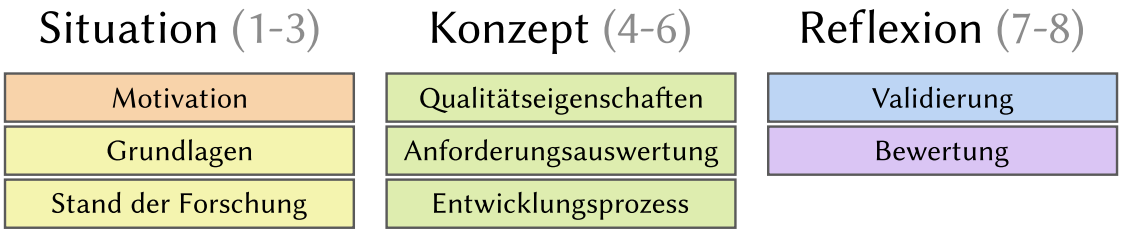

Abbildung 1.2: Dreiteilung der Kapitel dieser Arbeit in: Beschreibung der Ausgangssituation, Vorstellung der Konzepte und Reflexion der Ergebnisse

In Kapitel 2 werden die Grundlagen zum Mashup-Entwicklungsparadigma vorgestellt. Außerdem führt es die in der gesamten Arbeit genutzten Beispielszenarien und daran beteiligte Rollen ein. Schließlich wird eine Definition bzw. Interpretation der wichtigsten Begriffe der Qualität in Softwareprodukten vorgenommen. Kapitel 3 gibt einen Überblick bestehender Arbeiten zur Modellierung und Nutzung von Qualitätseigenschaften und -anforderungen in Web-Mashups bzw. verwandten Ansätzen. Daneben wird der Bezug zu einschlägigen Standards sowie vorherrschenden Entwicklungsparadigmen hergestellt. Kapitel 4 beschreibt die Struktur und Charakteristika des Qualitätseigenschaftsmodells für komposite Web-Mashups. Neben dem Metamodell enthält es ein Referenzmodell der für den Anwendungstyp charakteristischen Eigenschaften. Zur besseren Einordnung der gesamten Modellierungslandschaft werden die Beziehungen der Modelle dieser Arbeit in einem Überblick beschrieben. Kapitel 5 stellt das Konzept der vielseitig einsetzbaren Qualitätsanforderungen mit Fokus auf deren Spezifizierung und Auswertung zur Laufzeit und zur Entwicklungszeit vor. Hierbei wird besonderer Wert auf das nutzerzentrierte Festlegen von Anforderungen mit Assistenzwerkzeugen gelegt. Kapitel 6 gibt einen Überblick typischer Adaptionsaktionen sowie auf deren Integration in Qualitätsanforderungen und angepasste Entwicklungsprozesse. Besonders die Verbesserungen im Kontinuum der Entwicklungs- und Nutzungsprozesse, die durch die Einführung der anforderungsgetriebenen Komposition entstehen, werden ausgearbeitet. Kapitel 7 präsentiert die Konzepte und Ergebnisse in ihrer praktischen Umsetzung. Gleichzeitig werden die Validierungsmethoden und -ergebnisse vorgestellt, die die Belastbarkeit und Eignung der Konzepte innerhalb typischer Szenarien zeigt. Außerdem stellt das Kapitel einzelne Einschätzungen von Nutzern und Experten sowie Komplexitätsbetrachtungen und Benchmarks vor. Kapitel 8 fasst die gesamte Arbeit zusammen und führt eine abschließende Diskussion bzw. Bewertung durch. Schließlich werden in einem Ausblick relevante Anknüpfungspunkte an die Ergebnisse dieser Forschungsarbeit dargestellt.

Kapitel 2

Grundlagen der Entwicklung und Nutzung kompositer Web-Mashups

Charakteristisches Merkmal kompositer Web-Mashups ist die Verknüpfung webbasierter Ressourcen und Dienste zum Erstellen individueller Anwendungen. Es handelt sich dabei um Bestandteile mit und ohne Beitrag zum User Interface. Die Art dieser Verknüpfung sowie die strukturellen Eigenschaften der zu komponierenden Bestandteile bestimmen maßgeblich den Umgang während der Entwicklung und Nutzung der kompositen Anwendung als resultierendes Software-Artefakt. Sie begründen daher wesentliche Unterscheidungskriterien zu verwandten Forschungsfeldern. Daher konkretisiert dieses Kapitel die Rahmenbedingungen, unter denen komposite Web-Mashups im Zuge der Definition und des Einsatzes von Modellen für Qualitätseigenschaften und -anforderungen betrachtet werden. Zunächst wird eine Einordnung in komponentenorientierte Systeme sowie serviceorientierte Architekturen vorgenommen. Bezüglich ihrer Zielgruppen, ihrer Architektur und ihres Entwicklungsparadigmas werden anschließend unterschiedliche Ausprägungen von Mashups gegeneinander abgegrenzt. Berücksichtigt werden dabei repräsentative Anwendungsszenarien, anhand derer die universelle Einsatzfähigkeit der in dieser Arbeit vorgestellten Konzepte gezeigt wird. Nach der Vorstellung des erweiterten Rollenmodells, welches das Kontinuum der Entwicklungs- und Nutzungsprozesse verdeutlicht, wird schließlich eine grundlegende Definition und Abgrenzung der zentralen Qualitätsbegriffe und ihrer Zusammenhänge als Vorbereitung der Analyse existierender Arbeiten gegeben.

2.1 Charakteristika und Einordnung des Anwendungstyps

Die Grundeigenschaft von Web-Anwendungen im Allgemeinen ist der Einsatz von Web-Protokollen, -sprachen und -werkzeugen die in aktuellen Web-Browsern als Standardtechnologien verwendet und verarbeitet werden können. Es handelt sich dabei um das Hypertext Transfer Protocol (HTTP), die Hypertext Markup Language (HTML), deren Sprache zur Programmierung von Anwendungslogik JavaScript und Universal Resource Identifiers (URIs) als Adressierungssystem. Ergänzende Plugin-Technologien, beispielsweise Java Applets, Adobe Flash und Microsoft Silverlight, die aktuell noch für einzelne Anwendungsfälle – wie Dienste für Video on Demand (VoD) oder Browser-Spiele – genutzt werden, sehen sich zunehmend einer Verdrängung durch die zuvor genannten Standardtechnologien der Web-Browser ausgesetzt. Ihrem Einsatz zur Erbringung von Anwendungslogik in Black-Box-Anwendungsbestandteilen steht allerdings im Kontext dieser Arbeit nichts im Wege, jedoch werden die Spezifika ihrer Verwendung nicht besonders berücksichtigt. Der hier zugrundeliegende Anwendungstyp der *kompo-*

siten Web-Mashups zeigt außerdem Eigenschaften verschiedener verbreiteter architektonischer Paradigmen. Insbesondere sind dies:

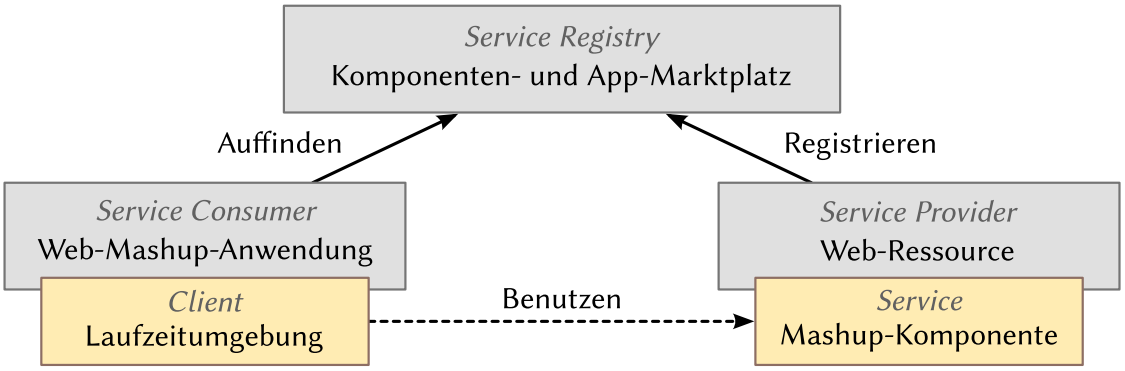

Abbildung 2.1: Projektion von SOA auf Web-Mashups

Serviceorientiertheit: Unter dem Paradigma der Serviceorientierte Architektur (SOA) können Web-Mashups und deren Bestandteile analog zu *Web-Services* betrachtet werden. In Abbildung 2.1 werden die kursiv gesetzten Begriffe und Rollen aus der SOA auf die Pendants einer Mashup-Infrastruktur projiziert. Dabei reicht die Art der erbrachten Dienstleistung von kleinen Aufgaben der Anwendungslogik, Berechnung oder Anbindung von Drittdiensten bis zur Bereitstellung von Benutzerschnittstellen als *User-Interface-Services*. In Abgrenzung zu Techniken der Web-Service-Komposition ergibt das Zusammenspiel der Dienste hier nicht einen neuen Web-Service, sondern eine Mashup-Anwendung, die einen Mehrwert bezüglich ihrer Fähigkeit zur Nutzerinteraktion und Anwendungslogik bietet. Mashup-Komponenten und auch größere Kompositionsfragmente werden in einem Marktplatz bzw. Verzeichnisdienst vorgehalten, der die Funktionen einer *Service Registry* übernimmt. Die Mashup-Anwendung – und aus ausführender Sicht die Laufzeitumgebung – erhält hierbei die Rolle des *Service Consumers*. Letztlich wird die Dienstleistung durch webbasierte Ressourcen als *Service Provider* erbracht, die in Form von Mashup-Komponenten im Marktplatz registriert sowie von der Anwendung genutzt werden können. Die so integrierten Anwendungsbestandteile setzen das Paradigma Software as a Service (SaaS) um. Eine Web-Anwendung, die sich SOA zur Integration von Anwendungsbestandteilen zunutze macht, heißt dann Serviceorientierte Web-Anwendung (SoWa). Die Art und Weise einer solchen dienstbasierten Auslieferung von Anwendungsbestandteilen ist eine wichtige Quelle für Qualitätseigenschaften, die später in einem Qualitätseigenschaftsmodell Berücksichtigung finden. Neben dieser generellen Implementierung des SOA-Paradigmas bestehen noch weitere kleine Abhängigkeiten, die dienstleistungsbasierte Architekturen in einer Mashup-Infrastruktur umsetzen. So kommt die Verwendung von Web-Ressourcen untereinander ggf. ohne eine Registry aus, erfüllt jedoch – etwa durch feste Adressierung von Web-Services – eine Dienstnutzung weitgehend nach dem vorgestellten Prinzip.

Komponentenorientiertheit: Anwendungen werden in funktionale Bestandteile zerlegt, die als Komponenten bestimmte Teilaufgaben wahrnehmen und durch Kommunikation die

Umsetzung einer komplexen Anwendungslogik erlauben. Zur Beschreibung der Schnittstellen ist das *Komponentenmodell* ein wichtiges Merkmal solcher Anwendungsbausteine. Hauptmotivation für eine einheitliche Komponentenbeschreibung ist die universelle Kombinierbarkeit der Bestandteile, die zu einer Vielfalt der Einsatzmöglichkeiten durch Wiederverwendung und damit zur Kostenreduktion in der Entwicklung und Wartung von Softwaresystemen führt. Im Einzelnen sind die zahlreichen Vorteile des Component-Based Software Engineering (CBSE) in [Szy02] beschrieben. Bei Mashups erfolgt dabei weniger die nachträgliche Aufteilung bzw. Dekomposition vorhandener Anwendungen zur besseren Wiederverwendung in weiteren Anwendungsfällen, vielmehr werden funktionale Einheiten absichtlich als Komponenten vorproduziert und erzeugen unterschiedliche Mehrwerte in verschiedenen Einsatzszenarien. Durch Integration der von Mashup-Bausteinen bereitgestellten Dienstleistungen als Komponenten, ergibt sich eine *Dualität von Service und Komponente* innerhalb der Anwendungsbausteine, von der der in dieser Arbeit betrachtete Anwendungstyp profitiert.

Anwendungsinterne Kommunikation: Im Gegensatz zu Portalanwendungen bzw. zu der von Dashboards bekannten parallelen Integration unabhängiger Widgets trägt die anwendungsinterne Kommunikation bei Mashups – d. h. der Informationsaustausch zwischen Mashup-Komponenten – substanziell zur Schaffung des Mehrwertes der Anwendung bei. Selbst mit einfachsten Kommunikationsparadigmen, wie dem unidirektionalen Publish-Subscribe-Muster, lässt sich eine komplexe Anwendungslogik aufbauen. Eine derartige Verbindung ermöglicht nicht nur die Kommunikation zwischen Bestandteilen des User Interface (UI), sondern auch den potenziell auf mehrere Laufzeitumgebungen verteilten Datenaustausch zu Service-Komponenten, die ggf. Anwendungslogik und Drittanbieterdienste bereitstellen.

Anwendungsbestandteile mit Nutzerschnittstelle: Die Gesamtoberfläche kompositer Anwendungen zeichnet sich meist durch visuell abgegrenzte Bereiche aus, die von UI-Komponenten bereitgestellt werden. Erfolgt die Bereitstellung dieser im Sinne einer Dienstleistung, vgl. oben, so entstehen *User-Interface-Services*, die im Zusammenhang kompositer Web-Anwendungen innerhalb der Arbeit von Pietschmann [Pie12] grundlegend eingeführt und betrachtet werden. Unter Beteiligung solcher UI-Services gewonnene UI-Mashups sind von Daten-Mashups abzugrenzen, die wie etwa *Yahoo! Pipes* ausschließlich Datenströme aggregieren, filtern und erneut Datenströme oder Datenfeeds anbieten und somit für einen Web-Nutzer nicht ohne Weiteres als Anwendung nutzbar sind. Analog verhält es sich mit kompositen Web-Services.

Situative Entwicklung: Vor allem die zuvor erwähnten Daten-Mashups sind aus situativen Problemen heraus entstanden, die keine klassischen Anwendungen mit Benutzeroberfläche produzieren, jedoch mit geringem Aufwand Datenströme und Feeds aggregieren und filtern. Mittlerweile wird angestrebt, dass die situative Entwicklung solcher Anwendungen, die vom späteren Nutzer getrieben wird, keine Expertenkenntnisse der Programmierung und Modellierung mehr voraussetzt. Stattdessen bieten Plattformen des EUD eine integrierte Entwicklung für Web-Nutzer als Programmierlaien an, um mittels Empfehlungsverfahren die vereinfachte Erstellung und Erweiterung – d. h. die *Live-Sophistication* – von Anwendungen zu ermöglichen. Das Fördern dieser Entwicklungsparadigmen durch

die geeignete Integration von Qualitätsmodellen und einer automatisierten Auswertungsumgebung ist ein zentrales Ziel dieser Arbeit, dessen entsprechende Konzepte als *Prozesskontinuum* in Kapitel 6 vorgestellt werden.

Eignung für Mobilgeräte: Erfolgreiche Anwendungsinfrastrukturen kommen in vielen modernen Nutzungsszenarien nicht mehr ohne eine umfassende Unterstützung mobiler Geräte aus. Zu berücksichtigende Besonderheiten bestehen bei der Umsetzung von Anwendungen und Komponenten vor allem im Layout (Platzbedarf, Display-Ausrichtung), in der Konnektivität (Caching, Offline-Fähigkeit) und in der Interaktionsfähigkeit (Tippen statt Klicken, Nutzung erweiterter Sensorik). Technologische Herausforderungen und deren Auswirkungen auf die Komplexität des Entwicklungsprozesses werden bei mobilen Web-Anwendungen durch abstrahierende Browserlaufzeitumgebungen, wie Apache Cordova [Apa15] betrachtet und führen so zur Entkopplung der inhaltlichen Entwicklung. Selbst die Erstellung nativer Apps für die jeweilige Zielplattform fußt damit auf webbasierten Entwicklungsparadigmen. So setzen beispielsweise Chrome Web-Apps auf die W3C-Recommendation *Packaged Web Apps* (Widgets) [Widg12]. Für den Windows Store gibt es die Möglichkeit der Entwicklung von Apps ausschließlich mit JavaScript und HTML [MS14]. Die Berücksichtigung dieser – vor allem für den mobilen Einsatz von Web-Anwendungen – relevanten Rahmenbedingungen ist eine wichtige Quelle für Qualitätseigenschaften, die sowohl in nutzerdefinierten Anforderungen als auch in automatisierten Regelkreisen genutzt werden können.

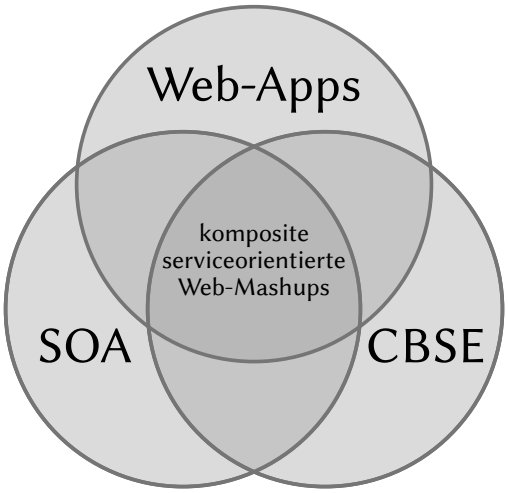

Abbildung 2.2: Zu Web-Mashups verwandte Architektur- und Anwendungstypen

Abbildung 2.2 zeigt eine Übersicht der Unterschiede der vorgestellten verwandten Anwendungstypen und deren Überschneidung. Diese Arbeit konzentriert sich auf UI-basierte, serviceorientierte, komposite Web-Mashups, d. h. komposite Anwendungen, die webbasierte Ressourcen als Dienstleistung integrieren und durch deren intelligente Verknüpfung einen Anwendungsmehrwert erzeugen. Die in der verbleibenden Arbeit vorgestellten Konzepte, Modelle und Anwendungsszenarien für anforderungsgetriebenes Qualitätsmanagement unterstellen grundsätzlich diesen Anwendungstyp als Rahmenbedingung.

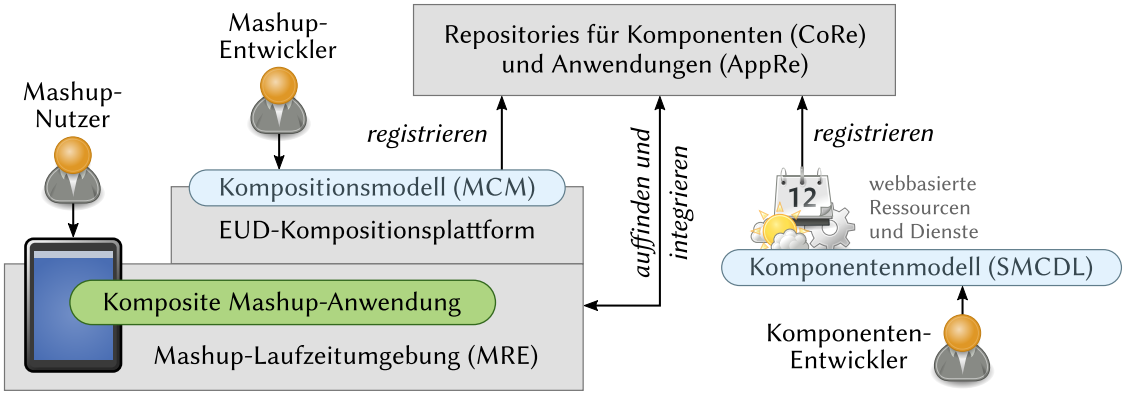

Abbildung 2.3: Überblick der Referenzinfrastruktur für komposite Web-Mashups

In [Pie12, Unterabschnitt 4.3.2] wird eine Referenzarchitektur für komposite Web-Mashups vorgestellt, an der sich grundsätzlich das in dieser Arbeit präsentierte System mit seiner Infrastruktur der Modellierung und Laufzeit orientiert. Die daraus entstandene Plattform trägt mittlerweile das Label Composition of Rich User Interface Services for Everybody (CRUISE) in Anlehnung an das Forschungsprojekt Composition of Rich User Interface Services (CRUISe), vgl. [Pie+09]. Einen Überblick der wichtigsten Bestandteile dieser Mashup-Infrastruktur zeigt Abbildung 2.3. Neben der Laufzeitumgebung *MRE* bildet das *Component Repository (CoRe)* den Kern der Ausführungsplattform. Dabei erfolgt das Auffinden und die Verwaltung der Mashup-Komponenten über ihre modellhafte Beschreibung in der SMCDL. Analog wird ein Anwendungsrepository angeboten, das als Marktplatz für komposite Mashups genutzt wird und deren Anwendungsbeschreibung in Form des Mashup Composition Model (MCM) ausliefert. Dieser Marktplatz kann analog zu *Stores* für Anwendungen gesehen werden, wie sie aktuell in vor allem mobilen Betriebssystemen angeboten werden, um deren Bewerbung und den ggf. kostenpflichtigen Bezug zu ermöglichen. Die MRE führt das Web-Mashup mit dem Ziel der Bereitstellung für einen Nutzer auf einem ihm zur Verfügung stehenden *Gerät* durch Interpretation des MCM aus. Dabei werden Mashup-Komponenten als Anwendungsbestandteile über das CoRe in das komposite Mashup integriert. Die tatsächlichen Web-Ressourcen – wie etwa Script-Dateien, Bilder und Web-Services – sind innerhalb der SMCDL referenziert und können beliebig extern gehostet sein. Die Wartung der Mashup-Komponenten und deren Ressourcen übernimmt dabei der Komponentenentwickler, der sie als Black-Box-Bestandteile anbietet. Derartige Kompositionsfragmente – sowohl auf Anwendungs- als auch auf Komponentenebene – sind als *semantische Modelle* repräsentiert bzw. werden in solche automatisch überführt. Die dafür zugrundeliegenden Technologien RDF und OWL zur Wissensrepräsentation sowie SPARQL für Abfrage und Austausch ermöglichen den Einsatz etablierter Frameworks und Produkte. Außerdem können hilfreiche Modelle von Drittanbietern, wie etwa *QUDT* [Hod+14] oder Domänenmodelle für Aktivitäten und fortgeschrittene Techniken wie Reasoning – beispielsweise für das Berechnen von Vorschlägen – genutzt werden. Eine umfassende Einführung zur Nutzung dieser grundlegenden semantischen Technologien bietet [Hit+08]. Im Sinne der integrierten Entwicklung situativer Mashups ergänzt eine *Kompositionsplattform* die Laufzeitumgebung, um unter anderem Live-Sophistication mit engen Iterationszyklen zu ermöglichen. Sie ist als Werkzeug eine

mögliche Form der Erzeugung neuer Anwendungsmodelle, wird durch den Mashup-Entwickler genutzt und befüllt das Anwendungsrepository mit MCMs. Die damit verbundenen Möglichkeiten und Herausforderungen des End User Development (EUD) in der Plattform CRUISE wurden in [Rüm+11] im Rahmen des Forschungsprojektes EDYRA untersucht.

2.2 Entwicklungsmethoden und Anwendungsszenarien

Neben den soeben vorgestellten *strukturellen Charakteristika* kompositer Web-Mashups, sind für das Anwendungsparadigma die *Eigenschaften des Entwicklungsprozesses* und *typische Szenarien der Nutzung* von großer Bedeutung. In dieser Arbeit werden Einsatzszenarien für Qualitätsanforderungen sowohl während der Entwicklungszeit als auch zur Laufzeit untersucht. Ziel dieses Abschnittes ist es, das Verständnis und den Formalisierungsgrad solcher Anforderungen über ggf. historisch gewachsene Entwicklungsprozesse hinweg aufzuzeigen und die fortschreitende Relevanz in aktuellen Entwicklungsparadigmen und -werkzeugen zu unterstreichen.

2.2.1 Entwicklungsmethoden für Web-Mashups

Die folgende Unterscheidung typischer Entwicklungsmethoden grenzt insbesondere die unterschiedlichen Grade der Automatisierung einzelner Vorgänge, die Einbeziehung spezialisierter Werkzeuge und den Grad des Explizierens von Anforderungen des Auftraggebers bzw. späteren Nutzers an die gewünschte Anwendung ab.

Manuell-situativ: Ursprünglich war die Mashup-Erstellung vom situativen Bedarf an geeigneten Werkzeugen für eine spezifische Aufgabe getrieben. Die Anfänge liegen dabei in der Ad-Hoc-Erstellung von Shell-Skripten – zum Zusammenfügen und Filtern von Datenströmen – sowie im Web-Bereich in der unstrukturierten Aggregation von Web-Ressourcen zu einfachen Anwendungen mit Portalcharakter. Dabei wurden die tatsächlichen *Anforderungen* an die erwünschte Anwendungsfunktionalität und an die erbringenden Software-Artefakte nicht explizit formuliert. Dieses sehr unstrukturierte Entwicklungsvorgehen kommt daher ohne spezielle Werkzeuge wie *Mashup-Enabler* – zur Erfassung bzw. Transformation von Anwendungsbausteinen in ein Komponentenmodell – und *Mashup-Builder* – zum Zusammenfügen dieser Bestandteile – aus. Es erfolgt keine Unterteilung in Entwicklungsphasen. Für viele situative Aufgaben ist dieses Vorgehen noch heute eine präferierte Option, da keine spezialisierte Ausführungsumgebung oder Toolchain zur Erzeugung einer lauffähigen Anwendung benötigt wird.

Technologiespezifische Adapter: Als nächsthöhere Stufe der Strukturierung im Entwicklungsprozess sind Plattformen für *Daten-Mashups* – wie Yahoo! Pipes [Yah11] – anzusehen, die in der Lage sind, einfache Feed-Formate zu aggregieren, zu filtern und selbst einen Datenstrom bzw. einen Feed als Ergebnis des Daten-Mashups anzubieten. Eine solche Nutzbarmachung gleichartiger Ressourcen mit technologiespezifischen Adaptern ist über gebräuchliche Datenformate wie RSS hinaus auch für Containerformate wie HTML Packaged Web Applications (W3C Widgets) geeignet. Nicht berücksichtigt ist hier zunächst die Art und Weise der Bereitstellung, da meist kein entsprechendes Service-Konzept beschrieben wird, vgl. Abschnitt 3.1. Prinzipbedingt wird in einer solchen Infrastruktur nur

genau das durch die Adapter nutzbar gemachte Komponentenmodell oder Containerformat unterstützt. Der Mashup-Enabler ist somit implizit vorhanden und muss aufgrund der automatisierten Adaption nicht extra als Werkzeug angestoßen werden. Anforderungen können in expliziter Form nur sehr eingeschränkt, beispielsweise über Namenskonventionen in den Metadaten der jeweiligen Formate oder Schnittstellenbeschreibungen, weitergegeben werden.

Mashup-Enabling durch Scraping: Grundsätzlich unstrukturierte Datenquellen beliebiger Web-Anwendungen können mit dem besonderen Einsatz eines Mashup-Enablers in eine von der Mashup-Plattform nutzbare Form gebracht werden. Hierfür existieren Scraping-Ansätze, die auf den unstrukturierten Datenressourcen ein für die Plattform nutzbares Format aufbauen. Eine verwandte Vorgehensweise erzeugt durch die Technik *Streamlining* über Scripting-Frameworks wie Greasemonkey fertige Anwendungen, ohne den Zwischenschritt über Mashup-Komponenten zu gehen, vgl. [DAI10]. Ein häufiger Anwendungsfall ist die Nutzung von Scraping-Technologien zum halbautomatischen Überführen vorhandener Web-Seiten oder ihrer Bestandteile in Komponenten. Hierbei besteht bei längerfristigem Betrieb der damit erstellten Anwendungen das Erfordernis, die eingesetzten Parser auf sich ändernde Quellformate anzupassen. Darüber hinaus sieht die Nutzungslizenz der gewünschten Daten oft keine maschinelle Weiterverarbeitung, Reaggregation und öffentliche Verbreitung vor, weshalb die Nutzung durch Scraping erzeugter Inhalte vielfach Probleme bereitet.

Mashup-Komponentenmodell: Web-Mashups sind – gemäß dem zuvor abgegrenzten Anwendungstyp – aus Mashup-Komponenten, die über ein Repository verfügbar sind, aufgebaut. Jede dieser Komponenten wird in einem festgelegten Format bereitgestellt, das vor allem die Kommunikationsschnittstellen und die Modalitäten der Auslieferung bzw. der Integration spezifiziert. Es liegt in der Verantwortung des Komponentenautors, wie er das vereinbarte Komponentenmodell implementiert und ob er Generatorwerkzeuge zur Erschließung fremder Quellinhalte nutzt. Mashup-Enabler werden hierbei somit komplett aus der Betrachtung genommen. Mashup-Builder, können sich auf ein festgelegtes Modell konzentrieren. Qualitätseigenschaften, die zur Weitergabe formalisierter Anforderungen dienen, sind idealerweise Teil dieses Modells für Mashup-Komponenten.

Live-Sophistication: Durch intelligente Werkzeugunterstützung verschmelzen die Prozesse der Entwicklung und Nutzung zeitlich und räumlich. Insbesondere für situative Aufgaben beschleunigt die Integration von Entwicklungswerkzeugen in eine Ausführungsplattform die Erstellung gewünschter Anwendungsfunktionalität. Anforderungen an die zu erstellende Mashup-Anwendung können hierbei in engen Zyklen umgesetzt werden. Das Konzept der Live-Sophistication – der Verbesserung von Anwendungen während ihrer Benutzung – wird in [Rüm+11] vorgestellt.

Im Fokus dieser Arbeit steht ein modellbasierter Entwicklungsprozess, wobei das Modell auf Anwendungsebene die Komposition der Bausteine beschreibt. Zur Unterstützung der Erstellung bzw. Anreicherung von Mashup-Komponenten stehen im Idealfall Importfilter zur Verfügung, die neben der Anwendungslogik auch Qualitätseigenschaften aus anderen Formaten

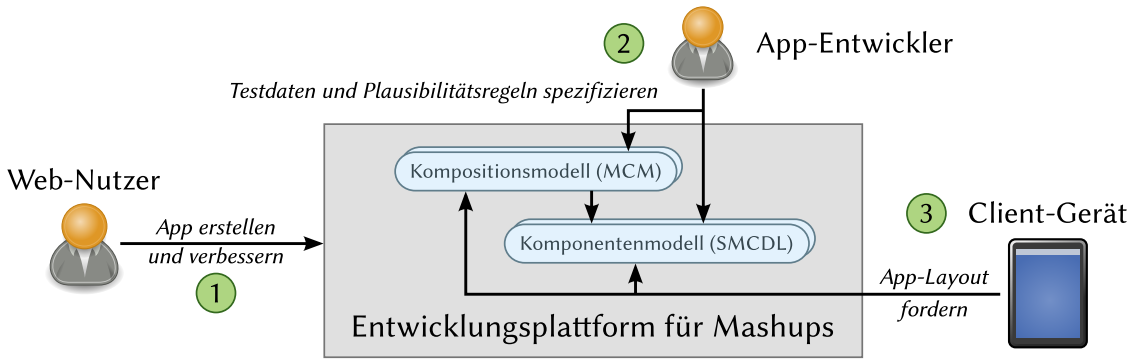

Abbildung 2.4: Szenarien der Entwicklung und Nutzung von Web-Mashups mit Qualitätsanforderungen über Komponenten- und Kompositionsmodellen

übernehmen oder übertragen können. Ein entscheidender Vorteil dabei ist, dass standardisierte bzw. andere verbreitete Formate somit automatisiert nutzbar gemacht werden können. Auf diese Weise kann zur Verbesserung des Entwicklungsprozesses beigetragen werden. Durch den hier angestrebten formalen Umgang mit Qualitätsanforderungen werden Entwicklungsprozesse gefördert, die in enger Kopplung von Entwicklungs- und Nutzungsaufgaben die situative Erstellung bzw. Verbesserung kompositer Web-Mashups ermöglichen. Nicht im Fokus dieser Arbeit steht der reine Vorgang des Mashup-Enablings – die Bereitstellung heterogener Ressourcen und Inhalte in einem einheitlichen, zur Komposition geeigneten Format. Ein zugrundeliegendes Mashup-Komponentenmodell wird als Rahmenbedingung unterstellt, vgl. dazu die Einordnung in die Referenzarchitektur in Abschnitt 2.1.

2.2.2 Szenarien der Nutzung und Entwicklung kompositer Web-Mashups

Die vorgestellten Methoden der Mashup-Erstellung sind als Phasen in den nun betrachteten Entwicklungs- und Nutzungsprozessen bzw. -szenarien verankert. Exemplarisch zeigt Abbildung 2.4 drei grundsätzlich verschiedene Szenarien der Entwicklung kompositer Web-Mashups unter Nutzung von Qualitätseigenschaften, die wesentliche Konzepte dieser Arbeit unterstützen sollen: ① Live-Sophistication mit Nutzung der Entwicklungsplattform als Black-Box zur Unterstützung des EUD, ② klassische Verwendung von Mashups als fertige Anwendungen, jedoch Nutzung der Vorteile des Entwicklungsprozesses bzw. des Kompositionsparadigmas der Web-Mashups und ③ automatischer Anforderungsaustausch und Selbstadaption ohne zwingende direkte Beteiligung von Nutzern oder Entwicklern.

Szenario ①: Integrierte Entwicklung, Verbesserung und Nutzung

Zentrale Rolle bei dieser Live-Sophistication nimmt der Web-Nutzer ein, der keine Kenntnisse in der klassischen Anwendungsmodellierung hat. Er bedient sich stattdessen der Möglichkeiten einer Plattform für EUD. Jeglicher Zugriff auf Modelle oder Programmcode wird dabei über das UI der Entwicklungsplattform als Black-Box angeboten. Web-Nutzer Anton möchte sich während einer Urlaubsreise über wichtige Sehenswürdigkeiten informieren, sie auf einer geographischen Karte anzeigen, Hintergründe dazu in Online-Enzyklopädien wie Wikipedia re-

cherchieren und sich ein Bild über die Wetterverhältnisse am aktuellen Ort machen. Neben üblichen Optimierungskriterien, wie etwa der *Antwortzeit*, dem *Preis* und der *Aktualität der Daten* von verwendeten Mashup-Komponenten, die in seiner Anwendung eingesetzt werden sollen, werden nutzerspezifische Qualitätsanforderungen gestellt. Anton legt Wert auf die Bewertung der Zufriedenheit anderer Nutzer. Er möchte gern *kostenlose* Komponenten einsetzen, die eine mindestens *gute* Zufriedenheitsbewertung haben. *Preisgünstige* Komponenten, die allerdings eine *exzellente* Zufriedenheitsbewertung haben, sind für ihn ebenfalls akzeptabel. Während der Entwicklung bzw. Verbesserung seiner Anwendung schaut er sich in Frage kommende Komponenten an und ist insbesondere am Energieverbrauch interessiert. Dabei fordert Anton, dass ein *möglichst niedriger* Energieverbrauch als Vorgabeanforderung für alle Auswahlprozesse von Komponenten gelten soll. Anschließend betrachtet er die ihm empfohlenen Vorschläge, welche innerhalb der Komposition die gewünschte Funktionalität liefern. Möglicherweise ist Anton noch nicht ganz zufrieden und nimmt Feineinstellungen vor, beispielsweise macht er die Bedingung über den Energieverbrauch restriktiver. Letztlich soll die Kompositionsplattform Vorschläge für Anwendungsbestandteile anbieten, die den soeben vorgestellten *Nutzeranforderungen* bestmöglich entsprechen. Anton wünscht sie nach erfolgter Bestätigung anschließend sofort in einer in die Laufzeitumgebung integrierten Form als Anwendungsfunktionalität zu nutzen. Idealerweise werden ihm einige der besten Treffer gemäß der Reihenfolge in der Bewertung zur Auswahl angeboten, welche zusätzlich textbasierte Suchparameter über die funktionalen Fähigkeiten bzw. weitere Metadaten berücksichtigt.

Szenario ②: Laufzeitüberprüfung mit vordefinierten Anforderungen

Klassische Anwendungsszenarien, die einen professionellen Entwickler bzw. Modellierer für Anwendungen einschließen, profitieren ebenfalls von einer qualitätsbewussten Mashup-Infrastruktur. Selbstverständlich können während der eigentlichen Anwendungsentwicklung zunächst die Vorteile der Live-Sophistication aus Szenario ① in Anspruch genommen werden. Primär wird in diesem Szenario jedoch eine fertig komponierte Anwendung unterstellt, die Grundlage für verschiedene Arten von *Laufzeitüberprüfungen* ist. So ist es beispielsweise möglich, Testdaten und *Plausibilitätsregeln* – wie etwa Wertebereichs- und Datenstrukturvalidierungen – zu spezifizieren, die zur Laufzeit gegen die Anwendungsdaten der Kommunikationsschnittstellen der Mashup-Komponenten geprüft werden.

Die professionelle Anwendungsentwicklerin Bettina stellt neben der von ihr gebauten kompositen Mashup-Anwendung einen Satz von Qualitätsanforderungen bereit, die zur Laufzeit gelten sollen. Zusätzlich zu den Bedingungen, deren Erfüllung es zu überprüfen gilt, muss sie sich in ihrer Rolle als Anforderungsautorin zweierlei zusätzliche Angaben überlegen. Einerseits wird spezifiziert, *zu welcher Gelegenheit* oder *wie oft* die Überprüfung stattfinden soll. Andererseits definiert sie, was geschehen soll bzw. welche *Aktionen* eingeleitet werden sollen, wenn die Anforderung während einer Überprüfung nicht erfüllt wird.

Die Anwendung beinhaltet eine Merkliste favorisierter Wertpapiere. Sobald ein bestimmtes Wertpapier ausgewählt wird, sollen dazu passende Wirtschaftsnachrichten angezeigt werden, beispielsweise Wirtschaftsdaten und Nachrichten zum Unternehmen. Die Mehrwertdaten werden von einem externen Anbieter über ein API bereitgestellt. Eine entsprechende Mashup-Komponente holt sie als Client ab und sorgt für die Visualisierung. Dabei kann es vorkommen, dass das angebundene API zeitweise nicht erreichbar ist. Diesen Fall berücksichtigt Bettina im

Vorfeld und liefert mit ihrer Anwendung folgende Qualitätsanforderung: Sofern bei Auswahl eines Wertpapierproduktes das verknüpfte API für Mehrwertdaten innerhalb von zwei Sekunden keine Antwort liefert, tausche den Anbieter gegen einen geeigneten aus und benachrichtige den Nutzer über diese Adaptionsaktion. Ein einfacher Assistent wie im Szenario ① reicht hierbei nicht aus. Bettina nutzt zur Spezifizierung der Anforderung einen erweiterten Assistenten, der auch die Angabe von Events (Gelegenheiten zum Auslösen der Auswertung) und Aktionen (Maßnahmen zur Gewährleistung der vorgesehenen Adaptivität) unterstützt.

Sobald die Anwendung mit den Qualitätsanforderungen erstellt wurde, findet sie Web-Nutzer Christoph im Anwendungsmarktplatz. Er entscheidet sich dafür, die von Bettina angebotene Mashup-Anwendung zu nutzen und kann in der Laufzeitplattform noch entscheiden, ob er die mitgelieferten Qualitätsüberprüfungen zur Laufzeit durchführen lassen möchte. Er bestätigt dies und tritt nun in die Nutzungsphase ein. Während der Nutzung tritt der Fall ein, dass die geforderte Dienstantwortzeit überschritten wird. Die entsprechende Qualitätsanforderung sieht vor, dass ein automatischer Austausch gegen eine vergleichbare Komponente erfolgen soll. Damit sich Christoph nicht wundert, warum die Mashup-Plattform eigenverantwortlich Veränderungen an der laufenden Anwendung vornimmt, setzt das System ihn mit einer kurzen Meldung über die Aktion in Kenntnis. Für Web-Nutzer Christoph bleibt die Anwendung somit trotz des Netzwerkproblems operabel, ohne dass er manuelle Veränderungen in der Anwendung vornehmen musste.

Szenario ③: Automatisierte Adaption

Der hier angestrebte hohe Formalisierungs- und Automatisierungsgrad von sowohl der Spezifikation der Qualitätseigenschaften und -anforderungen als auch der Anforderungsauswertung eröffnet zusätzlich Anwendungsfälle ohne zwingende Nutzerinteraktion. Sinnvolle Anwendungsfälle ergeben sich hierdurch in der vollautomatischen, anforderungsgetriebenen Auswahl von Anwendungsbestandteilen mittels der Qualitätseigenschaften, wie sie konzeptionell auch bei Web-Services bekannt ist. Sowohl bei der Anforderungsquelle als auch beim Anforderungsziel ist kein menschliches Eingreifen notwendig.

Web-Nutzerin Diana nutzt ein komposites Web-Mashup, das zuvor von ihr (Szenario ①) oder jemand anderem (Szenario ②) gebaut wurde. Die Laufzeitumgebung wird dabei auf ihrem mobilen Gerät ausgeführt, das begrenzten Bildschirmplatz zur Anzeige der Inhalte hat. Neben möglicherweise bereits geltenden Qualitätsanforderungen, die nutzergetrieben zustandegekommen sind, spielt in diesem Szenario eine weitere Art von Anforderungen eine Rolle: Das Client-Gerät – auf abstrahierter Betrachtungsebene die Betriebssystem-Plattform – hat Qualitätsanforderungen an das Layout der Anwendung und somit der Komponenten. Exemplarisch wird in diesem Szenario unterstellt, dass eine UI-Komponente zur Visualisierung von Slides einer Bildschirmpräsentation bestimmte Größenbeschränkungen bezüglich Höhe und Breite hat, um ihren Inhalt sinnvoll wiedergeben zu können. Setzt Diana nun ihr Gerät zur Ausführung der Anwendung ein, so ist es für sie primär unerheblich, welche Implementierung für die Präsentationskomponente die Mashup-Plattform einsetzt. Ihr Client-Gerät kann jedoch ergänzende Qualitätsanforderungen an die Komponenten stellen. Somit ist David als menschlicher Akteur in diesem Szenario nicht unmittelbar beteiligt. Die Anforderung wird automatisiert erstellt und innerhalb der Mashup-Infrastruktur an die verantwortliche Stelle zur Auswertung übergeben. Hier könnte als Aktion das Auslösen eines Alternativvorschlags ausgelöst werden, der eine für die Layoutan-

forderung besser passende Komponente empfiehlt. Ein mobiles Gerät generiert als Anzeigegerät für die Mashup-Anwendung vollautomatisch die Qualitätsanforderung über die Anzeigegröße bzw. die Anordnung der Komponenten und macht sie gegenüber der Laufzeitplattform bzw. dem Layoutmodell geltend. Erweiternd eignet sich dieses Vorgehen als Vorstufe zur Filterung in einem *Empfehlungssystem*, das dem Entwickler Assistenz im Erstellungsprozess bietet. Zur Laufzeit eröffnet sich zusätzliches Potenzial zur *Selbstadaption* der Anwendung, die auf plötzlich veränderte Rahmenbedingungen z. B. durch *Austausch* geeigneterer Anwendungsbestandteile oder mit *Feedback* an den Nutzer reagieren kann.

Aus diesen Szenarien ergeben sich spezifische Herausforderungen für die qualitätsbewusste Infrastruktur zur Entwicklung und Ausführung kompositer Web-Mashups sowie der begleitenden Modellierungslandschaft. Zum einen sollen durch eine hohe Selbstverantwortlichkeit der Plattform die verschiedenen Entwicklungsaufgaben unter menschlicher Beteiligung unterstützt werden. Insbesondere auf Qualitätseigenschaften basierende Empfehlungssysteme und kurze Iterationszyklen durch eine verzahnte Entwicklung und Nutzung im Zuge der Live-Sophistication wirken hier fördernd. Zum anderen ist es ein wichtiges Ziel, das Nutzungserlebnis durch die beschriebene Selbstadaption des Anwendungssystems und Feedback-Angebote zu steigern. Die vielseitigen Existenzformen der hier vorgeschlagenen Qualitätsanforderungen werden in Gestalt von Plausibilitätsregeln und Testdatensätzen ergänzt. Die Notwendigkeit der Ergänzung einer solchen automatisierten Überprüfung deckt sich mit der Analyse von Grammel und Storey: »The overall support for software engineering techniques such as testing and debugging in mashup development environments is quite limited.« [GS10].

2.3 Rollen im Entwicklungsprozess von Web-Mashups

Nach der Betrachtung der architektonischen Rahmenbedingungen und der geeigneten Entwicklungsmethoden werden nun die Rollen beschrieben, die an der *Entwicklung* und *Nutzung* kompositer Web-Mashups beteiligt sind. Die Entwicklung kann dabei in *Anwendungsentwicklung* – die in dieser Arbeit primär betrachtete Phase – und *Komponentenentwicklung* abgegrenzt werden, vgl. dazu auch [Wat07]. Ursprünglich – insbesondere bei einer situativen und unstrukturierten Entwicklung, aber auch teilweise bei der Live-Sophistication – sind bestimmte Rollen in einer Person vereinigt, die sich ggf. Zwischenschritte der Anforderungsdefinition oder Anwendungsmodellierung als formal abgegrenzte Aktivitäten erspart. Werden allerdings skalierbare Ressourcen, hohe Wiederverwendbarkeit, komplexe Anwendungsfunktionalität oder das Etablieren eines bestimmten Geschäftsmodells zur Nutzung – vgl. Platform as a Service (PaaS) – gewünscht, so differenzieren sich die Rollen stark. Neben der Unterteilung von Rollen nach ihrer Beteiligung in verschiedenen Phasen, kann auch zwischen menschlichen Akteuren und infrastrukturellen Bestandteilen der Plattform, wie etwa Kontextmonitore oder Repositorys, unterschieden werden. Zunächst wird auf die charakteristischen Merkmale jeder identifizierten Rolle eingegangen. Schließlich nimmt dieser Abschnitt eine Einordnung der vorgestellten Rollen in die Mashup-Infrastruktur vor und beleuchtet wichtige Situationen, in denen sich Rollen auf bestimmte Personen bzw. Infrastrukturbestandteile vereinigen.

2.3.1 Zur Entwicklungszeit

Komponentenentwickler: Der Komponentenentwickler (synonym auch: Komponentenautor) erstellt die komponierbaren Anwendungsbestandteile und stellt sie ggf. in einem Repository zur Verfügung. Bei der Entwicklung obliegt ihm dabei die Verantwortung, ob er Mashup-Komponenten für einen ganz bestimmten Einsatzzweck, zur Nutzung in einem bestimmten Anwendungskontext, oder sehr generisch ausstattet. Dies betrifft insbesondere die Ausarbeitung der Kommunikationsschnittstellen, die maßgeblich die Fähigkeit zur Kopplung der Komponenten in einer Anwendung bestimmen. Der Entwickler handelt dabei üblicherweise im Auftrag des Kunden bzw. späteren Nutzers oder stellt Komponenten für neue, in Mashup-Anwendungen zu nutzende Web-Ressourcen bereit. Als Artefakte produziert der Mashup-Komponentenentwickler das *Komponentenmodell* und kümmert sich um die Bereitstellung benötigter *Web-Ressourcen*, die ggf. extern gehostet werden. Das Komponentenmodell ist im Sinne dieser Arbeit vom Entwickler mit statischen Qualitätseigenschaftswerten und Metadaten versehen, die später zur Anforderungsauswertung verwendet werden können, vgl. Unterabschnitt 4.3.2 und Abschnitt 5.5.

Mashup-Entwickler: Unter Nutzung der in Komponenten-Repositorys verfügbaren Mashup-Komponenten ist es die Aufgabe des Anwendungsentwicklers, komposite Anwendungen zu erstellen. Neben der Auswahl geeigneter Mashup-Komponenten beinhaltet seine Verantwortlichkeit die Erstellung eines Kommunikationsflusses, der Layouts und ggf. eines Screenflows, d. h. aller wesentlichen Bestandteile eines Mashup-Kompositionsmodells, vgl. [Pie12]. Hierbei sind unterschiedlichste Automatisierungsgrade bezüglich der Erstellung der Bestandteile des Kompositionsmodells mit der Unterstützung von Empfehlungssystemen oder intelligenten Werkzeugen der Entwicklungsplattform möglich. Schon bei der Suche und Auswahl der Mashup-Komponenten ergänzt der Mashup-Entwickler seine Vorstellungen der funktionalen Eigenschaften der Zielanwendung durch *Qualitätsanforderungen* an in Frage kommende Kandidaten. Auf diese Weise kann er die Eignung für seinen individuell benötigten bzw. in Auftrag gegebenen Einsatzzweck quantifizieren. Analog zum Komponentenentwickler ist hier ebenfalls die Angabe von Qualitätseigenschaften auf Anwendungsebene möglich. Mittels geeigneter Werkzeugunterstützung der Entwicklungsplattform zum EUD wird es Entwicklern ohne weitreichende Programmier- bzw. Modellierungskenntnisse ermöglicht, die Rolle des Mashup-Entwicklers einzunehmen, vgl. Szenario ①.

Kunde: Die originäre Idee bzw. die Anforderungen an die Anwendung stammen vom Auftraggeber bzw. Kunden, der in einem situativen Mashup-Entwicklungsszenario die Anwendung später auch nutzt. Der Auftraggeber steht dabei in Dialog mit dem Requirements Engineer bzw. unmittelbar mit dem Entwickler der Anwendung. Aus softwaretechnischer Sicht wird dabei zunächst immer die direkt mit der Plattform interagierende Rolle betrachtet. Potenzial zur Differenzierung ergibt sich hierbei in der kollaborativen Nutzung eines Kompositionswerkzeuges durch beispielsweise den Kunden und den Requirements Engineer, um ggf. abweichende Anforderungen schnell anzugleichen.

Anforderungsexperte: Im verteilten Prozess der Anwendungsentwicklung hat der Anforderungsexperte oder *Requirements Engineer* die Aufgabe, Anforderungen, die originär durch

den Kunden bzw. Auftraggeber bestimmt werden, zu explizieren. Er tritt somit in Dialog mit diesen und den Entwicklern der Anwendung. In einer intelligenten Plattform zur Mashup-Entwicklung kann diese Rolle auch zum großen Teil durch Werkzeuge und Assistenten übernommen bzw. durch die formalisierte Übernahme der Anforderungen zwischen verschiedenen Entwicklungsphasen erleichtert werden. Zur Vereinheitlichung der Rolle des Requirements Engineers in Softwareentwicklungsprozessen vgl. auch [Rup11]. In Szenario ② fällt sie mit der des Anwendungsentwicklers zusammen.

Domänenexperte: Im Unternehmenseinsatz gelten zur Anforderungsgewinnung und deren Spezifikation Domänenexperten oder Fachgebietsexperten als Bindeglied zwischen Entwickler und Kunde. Sie kennen im Gegensatz zum Anwendungsentwickler und zum Anforderungsexperten die Begriffe und Vorgänge der Anwendungsdomäne sehr genau, verfügen jedoch im Allgemeinen nicht über die notwendige Programmiererfahrung, um eine Anwendung klassisch entwickeln zu können.

2.3.2 Zur Laufzeit

Ressourcenanbieter: Neben Repositorys für Mashup-Komponenten und der Entwicklungs- und Laufzeitplattform selbst müssen verschiedene Arten von Ressourcen, die innerhalb von Komponenten referenziert werden, verfügbar sein. Neben statischen Inhalten, wie JavaScript-Dateien, Bildern, CSS- und Textdateien sind dies auch Web-Services, die verteilt Anwendungsfunktionalität bereitstellen. Diese Anbieter sorgen für die Bereitstellung und Erreichbarkeit dieser Ressourcen. Hohe Verfügbarkeit erreichen Sie beispielsweise durch die Nutzung von CDNs.

Betreiber: Die Betreiber der Laufzeitplattform für Mashups – maßgeblich bestimmt durch Mashup-Enabler und Mashup-Builder – spielen zunächst für den Umgang mit Qualitätsanforderungen während der Nutzung eine untergeordnete Rolle. Jedoch lassen sich hier erweiterte Geschäftsmodelle mit der Abstraktion von Anforderungen aufbauen. So könnte der Plattformbetreiber beispielsweise vorgefertigte Qualitätsprofile kostenpflichtig als Dienstleistung anbieten.

Mashup-Nutzer: Nutzer verwenden die Mashup-Laufzeitplattform, um komposite Anwendungen auszuführen. Dabei können sie auch Qualitätsanforderungen zur Laufzeit empirisch überprüfen und ggf. Feedback zur Anwendungsnutzung geben. Bei Unzufriedenheit sind sie in der Lage, Verbesserungen oder Erweiterungen des Mashups beispielsweise über das Paradigma der Live-Sophistication anzustoßen. Die Adaptivität der Mashup-Plattform macht sich der Mashup-Nutzer über Empfehlungsmechanismen zu Nutze, die es ihm auch erlauben, Anwendungen und deren Bestandteile zu vergleichen und unter Zuhilfenahme von Anwendungsmarktplätzen zu bewerten.

Kontextmonitor und -dienst: Werte für Qualitätseigenschaften, die nicht statisch angegeben werden können, müssen zur Laufzeit ermittelt bzw. berechnet werden. Der Kontextdienst stellt dabei wichtige Messwerte der Sensorik bereit. Er steht in Dialog mit der Laufzeitumgebung und weiteren Komponenten der Infrastruktur, wie dem Repository für Anwendungen und deren Bestandteile. Dort können ggf. auch weiterführende Berechnungen

durchgeführt werden, wie etwa die Ermittlung von Durchschnittswerten bzw. Summen. Für die Verwaltung von und den Zugriff auf Profile, die individuelle Mashup-Nutzer charakterisieren, ist der Kontextdienst ebenfalls einsetzbar.

Empfehlungsdienst: Empfehlungen (englisch: *recommendations*) für Mashup-Komponenten oder auch für größere Kompositionsfragmente, die komplexe Anwendungslogik bereitstellen, sind einerseits zur Entwicklungszeit möglich, andererseits auch zur Laufzeit sinnvoll, um die Adaptivität der Plattform zu nutzen. So können beispielsweise zur Laufzeit veränderliche Qualitätseigenschaften als Auslöser für einen Komponentenaustausch oder weitere Aktionen genutzt werden.

2.3.3 Einordnung der Rollen und Zielgruppen in der Mashup-Plattform

Die grundlegenden Rollen der *Entwickler von Mashups und Mashup-Komponenten* sowie die des *Mashup-Nutzers* decken sich mit dem Modell aus [Pie12]. Sie sind ebenfalls in der Referenzinfrastruktur in Abbildung 2.3 berücksichtigt. Die bei Pietschmann noch erwähnte Rollenübertragung auf den Bereich der Entwicklung und Komposition von Web-Services wird hier nicht weiter betrachtet, da Mashup-Komponenten bereits als universelle Kompositionsbausteine berücksichtigt sind. Durch die vielfältigen Anwendungsfälle, die sich aus einem durch Qualitätsanforderungen getriebenen Entwicklungsprozess ergeben, rücken die weiteren, oben beschriebenen Rollen in den Vordergrund. Dabei erschließen sich insbesondere durch das Einbeziehen des Plattformbetreibers und des Ressourcenanbieters der Infrastrukturbestandteile neue Möglichkeiten für Geschäftsmodelle – etwa durch die Kopplung einer klassischen PaaS mit kostenpflichtigen Zusatzdienstleistungen über Qualitätsprofile angepasster Anforderungen.

Während der integrierten Entwicklung kommt es zu starken inhaltlichen und zeitlichen Überschneidungen der durch die vorgestellten Rollen ausgeübten Aufgaben. Während die Bereitstellung von Mashup-Komponenten als Black-Box-Bestandteile weitgehend separiert bleibt, vermischt sich die Ausübung der Rollen des Mashup-Entwicklers – einer typischen Rolle der Entwicklungszeit – und die des Mashup-Nutzers als Laufzeitrolle. Einige Aufgaben, wie die Konfiguration von Filtern für die Auswahl während des EUD, die klassischerweise dem Mashup-Entwickler zugeordnet sind, übernimmt perspektivisch die Mashup-Plattform, indem sie zusätzlich zur Kerninfrastruktur Empfehlungsdienste und Kontextanbieter bereitstellt. Folglich ergibt sich eine potenzielle Verbesserung der Nutzung im Szenario ② durch hilfreiche Assistenzfunktionen für die automatischen Tests und zur Überprüfung der Plausibilität der Daten in anwendungsinterner Kommunikation.

Selbst in der speziellen Definition der Zielgruppe potenzieller Web-Nutzer, die für Entwicklungsaufgaben mit einer Mashup-Plattform mit EUD-Fähigkeiten in Frage kommen, bleibt großer Spielraum der tatsächlichen Bereitschaft, bestimmte Komplexitäten in der Anwendungskomposition zu erreichen. Mashups waren im Sinne der softwaremäßigen Verknüpfung von Ressourcen schon immer eher *kleine* und vor allem *aus einer praktischen Situation entstandene* Gebilde. Auch einfache Shell-Skripte oder Datenfeed-Mashups wie Yahoo! Pipes tragen diesen Charakter. Aus Mangel an Entwicklungsumgebungen war die Nutzung für Personen im Programmierumfeld beschränkt. Mit dem Aufkommen von Mashup-Werkzeugen für Consumer wurde die Zielgruppe auf weitere Web-Nutzer vergrößert. Die Gemeinsamkeit in der Art der Nutzung blieb jedoch: Die Idee und alle Anforderungen, die für die Anwendung gelten sollten,

kommen von *einer Person*. Es gibt keine separaten Kunden, die in mehreren Iterationszyklen Anforderungen diktieren oder ändern. Die Person ist neben der Entwicklung auch für den Testprozess verantwortlich, indem sie die Funktionalität der selbst erstellten Anwendung auf die erdachten Ideen und Anforderungen überprüft, ohne sie in Dokumenten zu explizieren. Ziel dieser Arbeit ist es, sowohl das situative Ein-Personen-Szenario durch eine intelligente Plattform-Begleitung zu fördern als auch hoch ausdifferenzierte Rollen durch das Anbieten geeigneter Schnittstellen der Modelle und Repositorys sowie eine Infrastruktur für die komplementäre Nutzung zu ermöglichen. Es gilt außerdem herauszufinden, in welchem Maße Nutzer ohne fundierte Programmierkenntnisse mit den Werkzeugen einer EUD-Plattform Anforderungen über Qualitätseigenschaften ihrer Mashups spezifizieren und konfigurieren können. Die kollaborative Entwicklung von Mashup-Anwendungen wird in dieser Arbeit zunächst nicht betrachtet, sodass hierfür keine gesonderten Rollen definiert werden müssen. Allerdings kann bei den im Konzept vorgeschlagenen Aktionen die Unterstützung von Experten angefordert werden, die eine Schnittstelle zu kollaborativer Entwicklung anbieten.

2.4 Qualitätseigenschaften und -anforderungen im Kontext von Web-Mashups

Neben den Charakteristika der Architektur von Web-Mashup, den dazugehörigen Entwicklungsmethoden und den im Erstellungs- und Nutzungsprozess beteiligten Rollen sind *Qualitätseigenschaft*, *Qualitätsanforderung* und damit verwandte Begriffe der Modellierung und Laufzeit wichtige Grundlagen zum Verständnis dieser Arbeit. Der vorliegende Abschnitt gibt eine kurze Definition, Einordnung und Abgrenzung der Qualitätsbegriffe, die teilweise alltäglich in verschiedenen Situationen mit bestimmten Bedeutungsausprägungen gebräuchlich sind. Definitionen, die einschlägige Standards dazu vorhalten, werden in Abschnitt 3.1 sowie die Interpretationen einzelner verwandter Arbeiten in den darauffolgenden Abschnitten gegeben.

Qualität: Grundsätzlich wird damit die *Beschaffenheit* oder *Güte* eines Systems, Objektes oder Prozesses bewertet. Ein Hauptunterscheidungskriterium – insbesondere im Softwarebereich – ist, ob es sich um ein *Produkt* oder einen *Prozess* handelt. Diese Arbeit betrachtet mit ihren Modellen und Konzepten in erster Linie die *Produktqualität*. Die in der Betrachtung stehenden Produkte sind dabei vor allem Web-Anwendungen und deren Bestandteile in Form kompositer Web-Mashups und entsprechender Komponenten. Wichtig bei der Qualitätsbetrachtung ist die *Wahrnehmbarkeit*, die im Rahmen der Messung bzw. der Kontextsensorik für diese Arbeit eine wichtige Rolle zur Datenerfassung und zur Anforderungsauswertung spielt.

Qualitätseigenschaft: Um die Produktqualität für ein bestimmtes System oder eine Anwendung messen, angeben und überprüfen zu können, muss diese in messbare bzw. bewertbare Eigenschaften (englisch: *quality properties*) zerlegt werden. Diese Qualitätseigenschaften (synonym auch: Merkmale, Attribute, Kriterien oder Faktoren) werden in typischerweise in Qualitätsmodellen organisiert, siehe unten. Dabei werden in anderen Arbeiten teilweise mehrere dieser synonymen Begriffe zur Abgrenzung verschiedener Hierarchiestufen und Granularitäten genutzt. Die genauen Umstände und Rahmenbedingungen, die

zur Wertermittlung führen, werden durch die *Qualitätsmetrik* festgelegt. Um die Generalisierbarkeit der Konzepte, Prozesse und der Modellierungslandschaft zu unterstreichen, werden in dieser Arbeit auch klassische Metadaten wie Autorenangaben und Versionen als Qualitätseigenschaften eingeordnet, da der Umgang mit solchen Angaben – wie sich später in den Konzeptkapiteln zeigt – analog erfolgen kann.

Qualitätsmetrik: Zu einer Qualitätseigenschaft gibt die Metrik an, wie genau die Bewertung der unter Betrachtung stehenden Größe erfolgt. Dazu zählt die Angabe der Einheit, mit welchem Verfahren und unter welchen Bedingungen der Wert der Eigenschaft gemessen oder angegeben wird. Somit wird zur *Adressierbarkeit* einer Eigenschaft über die Qualitätsmetrik die *Messbarkeit* und *Vergleichbarkeit* ergänzt, indem beispielsweise ein gemeinsam genutztes Vokabular zur semantischen Beschreibung und zur Überführung verschiedener Werte spezifiziert wird.

Qualitätsanforderung: Qualitätseigenschaften werden durch Anforderungen, die in spezialisierter Form auch als *Bedürfnisse*, *Bedingungen* oder *Zusicherungen* auftreten können, nutzbar. Erst durch diese Qualitätsanforderungen entfaltet sich das praktische Potenzial von Qualitätseigenschaften in Entwicklungsprozessen, bei der facettierten Suche oder innerhalb von automatischen Überprüfungen zur Laufzeit. Sie ermöglichen beispielsweise die Selbstadaption kompositer Anwendungen durch den intelligenten Austausch ihrer Bestandteile. Eine solche Anforderung enthält immer eine Bedingung, die über die Wertebelegung einer Qualitätseigenschaft gebildet wird. Dabei können Anforderungen auch kombiniert werden, etwa um Kompromisse auszudrücken. Solche kompositen Qualitätsanforderungen stellen neue Herausforderungen an die Wichtung und die zweckgerechte Aggregation der Einzelwerte, auf die detailliert in Kapitel 5 eingegangen wird.

Qualitätsmodell: Dieser Begriff wird meist als Kurzform für *Qualitätseigenschaftsmodell* verwendet, er kann jedoch auch zur Beschreibung von Qualitätsanforderungen stehen. In dieser Arbeit wird damit in der Regel – wenn im unmittelbaren Kontext nicht abweichend angegeben – auf das Qualitätseigenschaftsmodell Bezug genommen. Unter diesem Oberbegriff legt ein *Metamodell* die abstrakte Struktur und Abhängigkeiten zu anderen Modellen fest. Ein *Referenzmodell* enthält konkret die im betrachteten System relevanten Eigenschaftstypen und deren Zusammenhänge. *Instanzmodelle* dienen dann zur Erfassung bzw. Persistierung der Wertebelegungen pro Kompositionsfragment, die über die in den Metriken definierten Verfahrensweisen und Kontextsensoren entstehen.

Die Zuordnung der in diesem Abschnitt vorgestellten Rollen zu den Modellen für Qualitätseigenschaften und -anforderungen – d. h. die Beschreibung der manipulierenden und lesenden Zugriffe – wird detailliert in Unterabschnitt 4.3.2 und folgenden Abschnitten sowie in Kapitel 5 vorgenommen. In bestimmten Fällen wird in anderen Arbeiten und Kontexten umgangssprachlich *Anforderung* anstelle von *Eigenschaft* verwendet – beispielsweise wenn eine Komponente eine Zusicherung der maximalen Antwortzeit als Eigenschaft besitzt. In dieser Arbeit wird bei konkreten Anwendungsfällen von einer klaren Trennung ausgegangen, die im Sinne der eindeutigen Zuordnung zu infrastrukturellen Entitäten und der Behandlung in den Entwicklungs- und Nutzungsszenarien unabdingbar ist. Im genannten Beispiel könnten Anforderungen des Nutzers oder der Anwendung an die Komponente über diese Zusicherung, die selbst als Anforderung der Komponente an einen zugrundeliegenden Web-Service gilt, gebildet werden.

Kapitel 3

Stand der Forschung und Technik

Während die Betrachtung der Modellierung, Überwachung und Auswertung von Qualitätsanforderungen im Bereich kompositer Web-Mashups ein noch junges Forschungsfeld ist, gibt es in der Entwicklung und bei der Ausführung verwandter Anwendungsparadigmen, vor allem in der Softwarekomposition und bei Web-Services, bereits detaillierte Untersuchungen und Ansätze. Die im Zusammenhang mit der Spezifikation allgemeiner Anforderungen an Softwaresysteme in vorherrschenden Entwicklungsprozessen erzeugten Artefakte beschreiben dabei Anforderungen an verschiedene Qualitätsaspekte des entstehenden Systems. Dieses Kapitel untersucht die Spezifikations- und Nutzungsformen solcher Artefakte sowie dazugehörige Prozesse und gibt eine Einschätzung über deren Einsatztauglichkeit für die Ziele dieser Arbeit.

Nachdem zunächst auf die Bedeutung von Normen und Standards zur Abbildung von Qualität in Softwaresystemen eingegangen wird, erfolgt eine Übersicht von Ansätzen zur Erstellung und Strukturierung von Qualitätsmodellen. Danach wird ein kurzer Überblick gegeben, welche Rolle semantische Aufgabenmodellierung und das Konzept der Fuzzy-Mengen samt bisher verfügbarer Werkzeugunterstützung für die Definition dekomponierbarer und unscharfer Qualitätsanforderungen spielen. Anschließend werden Ansätze zur Modellierung, Integration und Auswertung von Qualitätsanforderungen und den zugrunde liegenden Eigenschaftsmodellen in Web-Mashups und dazu verwandten System- bzw. Anwendungstypen gegenübergestellt. Der detaillierte Vergleich wird mittels bestimmter Bewertungskriterien systematisiert, die in einer Übersicht die besonderen Stärken und Schwächen repräsentativer Ansätze der untersuchten Strömungen aufzeigen. Sofern anwendbar, stehen insbesondere die Dekomponierbarkeit im Sinne einer dienstleistungsorientierten Architektur und der Benutzerschnittstellencharakter als wesentliche Charakteristika für den Umgang mit Web-Mashups im Vordergrund der Analyse. Schließlich wird nach der detaillierten Einschätzung ein Fazit gezogen, das insbesondere verdeutlicht, wie die Konzepte und Vorgehensweisen innerhalb des in dieser Arbeit zugrunde liegenden Entwicklungsprozesses für komposite Web-Mashups und für deren Betrieb in einer passenden Infrastruktur genutzt werden können.

3.1 Normen und Standards für Qualitätsmodelle bei Softwareprodukten

Bei der allgemeinen Betrachtung von Qualitätseigenschaften eines Systems kann zunächst zwischen der Qualität des Produktes – d. h. von Artefakten, die innerhalb des Systems verarbeitet werden bzw. entstehen – und den Prozessen, die für die Verarbeitung verantwortlich sind, unterschieden werden. Diese Arbeit betrachtet auf Modellierungsebene primär den erstgenann-

ten Fall – die *Produktqualität*. Für diese Art von Modellen existieren mehrere relevante Standardisierungsgremien. Die International Organization for Standardization (ISO) veröffentlicht ihre Normen als *International Standards*. Im Bereich Elektrotechnik und Elektronik geschieht dies oftmals gemeinsam mit der International Electrotechnical Commission (IEC). Europäische Normungsorganisationen übernehmen diese meist als Europäische Norm (EN) bzw. ratifizieren übersetzte Versionen, die das Deutsche Institut für Normung DIN als nationale Normen umsetzt. Diese sind an mehrgliedrigen Präfixen wie *DIN EN ISO* erkennbar. Maßgebend für Web-Technologien ist das World Wide Web Consortium (W3C), dessen Normen als *Recommendations* veröffentlicht werden. Dazu zählen auch Basistechnologien und Protokolle, wie das Resource Description Framework (RDF) oder HTTP.

In zahlreichen verwandten Arbeiten sind die nun vorgestellten Standards Ausgangspunkt oder zumindest eine Inspiration für die dort verwendeten oder entwickelten Qualitätsmodelle. So z. B. definiert die Norm ISO/IEC 9126 [ISO9126] ein Modell zur Sicherstellung der Qualität von Softwareprodukten. Die dort präsentierten Qualitätsmerkmale werden in die sechs Gruppen *Funktionalität*, *Zuverlässigkeit*, *Benutzbarkeit*, *Effizienz*, *Änderbarkeit* und *Übertragbarkeit* zusammengefasst. Dabei sind die jeweiligen Teilmerkmale als Vorschläge den Gruppen zugeordnet. *Interoperabilität* gilt dort beispielsweise als Teilmerkmal der Gruppe *Funktionalität*. Mittlerweile wurde diese Norm durch die neuere Serie ISO/IEC 25000 abgelöst. Das Qualitätsmodell im Speziellen wurde dabei in ISO/IEC 25010 [ISO25010] integriert. Viele Arbeiten verwenden jedoch noch das Modell der älteren Norm ISO/IEC 9126 als Grundlage, siehe z. B. die verwandten Arbeiten zu Qualitätsmodellen in Web-Mashups in Abschnitt 3.5. In ISO/IEC 25010 sind die Qualitätsmerkmale in zwei Modellen organisiert. Besonderes Augenmerk liegt hier auf dem *Quality in Use Model* zur Betrachtung der Qualität eines Softwaresystems aus Nutzersicht, das *Effektivität*, *Effizienz*, *Zufriedenheit*, *Risikofreiheit*, und *Kontextabdeckung* beinhaltet. Das hier ebenfalls enthaltene *Product Quality Model* umfasst *Funktionale Eignung*, *Performanz und Effizienz*, *Kompatibilität*, *Benutzerfreundlichkeit*, *Zuverlässigkeit*, *Sicherheit*, *Wartbarkeit* und *Portabilität*. Das Quality in Use Model steht besonders in existierenden Arbeiten zur Qualität in Web-Anwendungen im Fokus, vgl. Abschnitt 3.6. Die Modelle der Standards sind bewusst allgemein gehalten, um sie in verschiedensten Anwendungsbereichen als Richtlinie angepasst einsetzen zu können.

Das W3C – federführend für die Standardisierung von Web-Technologien, -Services und entsprechender -Anwendungen – bietet bisher keine Norm zur umfassenden Beschreibung von Qualitätseigenschaften in seiner fachlichen Domäne. In den vielfältigen Einzeltechnologien zu Web-Services werden allerdings Methoden – insbesondere zur Aushandlung von Qualitätsanforderungen – definiert, die sich beispielsweise auf Angaben der Web Services Description Language (WSDL) stützen, siehe weiter unten im Abschnitt. Eine Norm zur Strukturierung und Verpackung von Web-Anwendungen als *Widgets* sieht die Recommendation *Packaged Web Apps* [Widg12] vor. Im Forschungsprojekt OMELETTE, vgl. [Chu+13], wurde diese als Komponentenmodell interpretiert und über Erweiterungen – beispielsweise der Kommunikation zwischen Widgets – zum Einsatz innerhalb kompositer Web-Mashups verwendet. Eine Nutzung in Verbindung mit anforderungsgetriebenen Entwicklungsprozessen ist automatisiert ohne Weiteres nicht möglich, da die Modellierungsgrundlage für Qualitätseigenschaften fehlt.

Die Standards zur Modellierung und Strukturierung von Qualitätseigenschaften achten konsequent auf die – in oberflächlicher Betrachtung meist vernachlässigte – Trennung zwischen *Eigenschaft* und *Anforderung*. Folglich existieren auch Normen zur Beschreibung von Quali-

tätsanforderungen sowohl für Softwareprodukte im Allgemeinen durch die Normenreihe ISO 250xx als auch im Web-Bereich in Form der WS-Policys. [ISO25030] definiert ein umfassendes Modell für Anforderungstypen. Der bei Web-Mashups typische Prozess der integrierten Nutzung und Entwicklung im Sinne der *Live-Sophistication* tritt nicht als betrachtete Zielphase auf, da hier ein klassischer Softwareentwicklungsprozess im Fokus steht, in dem Anforderungen in der *Spezifikation*, *Planung*, *Entwicklung* und *Evaluation* – beispielsweise im Rahmen einer Zertifizierung – in Erscheinung treten. Abbildung 3.1 zeigt die in der Norm vorgestellten Typen von Qualitätsanforderungen für Softwareprodukte. Die in dieser Arbeit angestrebten Anforderungen betreffen primär inhärente Produkteigenschaften, wobei sich die funktionalen Teile an Schnittstellenbeschreibungen und deren semantische Capabilitys richten. Bei der Softwarequalität kann insbesondere die Auswertung externer Qualitätsanforderungen durch Kontextsensorik erreicht werden. Der Vergleich interner Eigenschaften mit entsprechenden Anforderungen – beispielsweise auf Quellcode-Ebene der Komponentenimplementierung – ist aufgrund des Black-Box-Charakters nur sehr eingeschränkt vorgesehen. Man ist hier auf den Zugang über die Komponentenschnittstellen angewiesen, der explizit geschaffen werden muss. Quality in Use lässt sich üblicherweise durch vorher erfasste und statistisch erhobene Nutzerbewertungen evaluieren.

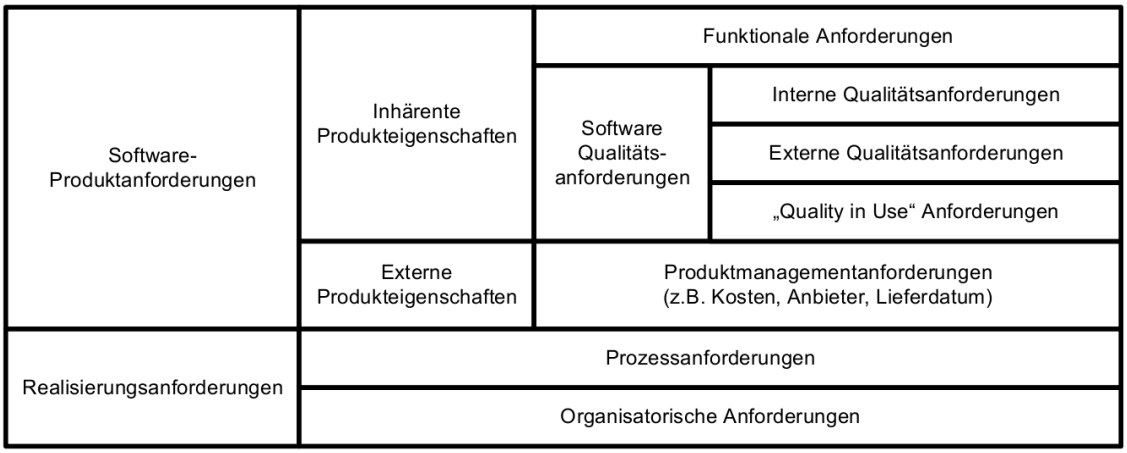

Abbildung 3.1: Typen von Qualitätsanforderungen nach [ISO25030]

WS-Policy [WSPol07] als Spezifikation des W3C erlaubt die Definition von Richtlinien zu Qualität und Sicherheit von Web-Services. Sie ist damit eine Möglichkeit zur Angabe von Qualitätsanforderungen, die bei Web-Services – insbesondere zur automatischen Aushandlung von Qualitätsparametern zwischen mehreren Web-Services – genutzt wird. Zweck der damit formulierbaren *Policys* ist dabei hauptsächlich die Aushandlung von Dienstausführungsbedingungen in XML zum Einfügen in WSDL-Beschreibungen. Sowohl *Mindestrichtlinien des Services* als auch die *Anforderungen des Servicenutzers* an den Service können mit WS-Policy ausgedrückt werden. Die Nutzungsarten der Policys *Required*, *Rejected*, *Optional*, *Observed* und *Ignored* werden als *Usage*-Attribute spezifiziert. Beispiel für eine WS-Policy ist die verschlüsselte Kommunikation mit dem Web-Service, indem mehrere Algorithmen als Alternativen angeboten werden, unter denen gewählt werden muss. *Assertions* beschreiben Standardrichtlinien zur Verwendung in einer Policy. Obwohl sich mehrere Parteien in einer Web-Service-Kommunikation auf *effektive*

Policies einigen können, ist nicht definiert, wie die semantische Auswertung der ausgehandelten Alternative erfolgt. Für die Nutzbarkeit in einer Mashup-Plattform mit automatischem Auslösen von adaptivem Verhalten bleibt der Ansatz WS-Policy nur eine Inspiration für das Definieren von Handlungsvorschriften, die in Kapitel 6 dieser Arbeit vorgestellt werden.

Insgesamt existieren zur Beschreibung der Produktqualität auch für Softwaresysteme nutzbare Vorgaben für Modelle von Qualitätseigenschaften. In den genannten Normen sind diese bewusst allgemein gehalten. Sie müssen auf System- bzw. Anwendungstypen abgestimmt werden. Diese Art von Anpassungen nutzen einige der in diesem Kapitel vorgestellten konkreten Ansätze für ihren jeweiligen Typ von Anwendung bzw. System. Web-Standards für komposite Anwendungen berücksichtigen Qualitätseigenschaften dieser Modelle jedoch gar nicht und nehmen auch keinen Bezug auf existierende Normen anderer Organisationen. Lediglich im Bereich der Web-Services wurden aufgrund des hohen Automatisierungsbedarfes bereits Konzepte der Aushandlung von Zusicherungen als Anforderungen in Form von Standards spezifiziert.

3.2 Strukturierung und Erstellung von Qualitätsmodellen

Zum Aufbau und zur Strukturierung der Qualitätseigenschaften in einem Modell stehen verschiedene Techniken zur Verfügung. Bei der Betrachtung vorhandener Ansätze liegt insbesondere die *Eignung* für komposite Web-Mashups als Anwendungstyp und die *Art der Verwendung* innerhalb der möglichst automatisierten Spezifikation und Auswertung von Qualitätseigenschaften im Fokus. Eine einfache Möglichkeit zur Strukturierung von Qualitätsmodellen folgt dem Ansatz Factor-Criteria-Metrics (FCM). Dort wird der Qualitätsbegriff in Eigenschaften unterteilt, die jeweils in Teileigenschaften zerfallen, vgl. Abbildung 3.2.

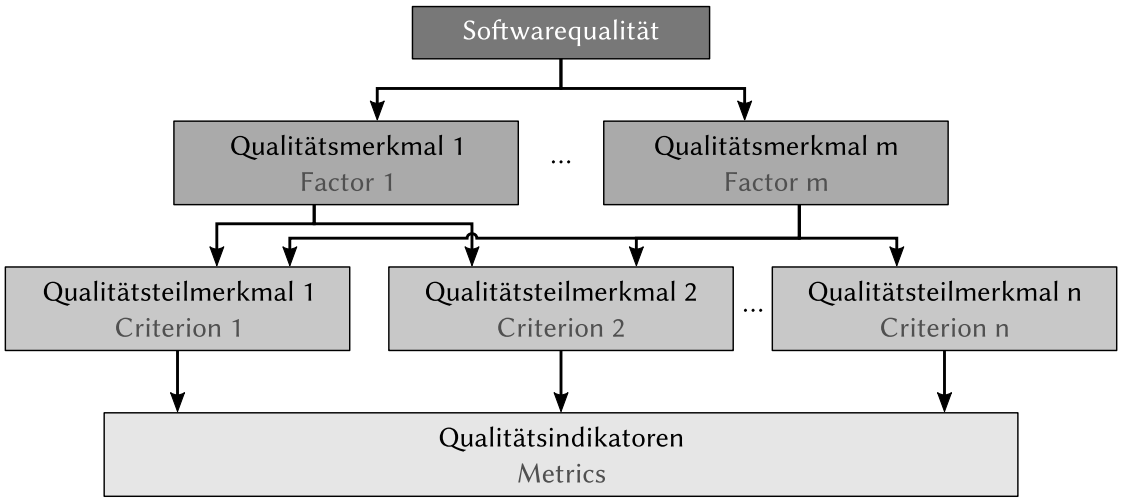

Abbildung 3.2: Strukturierung von Qualitätsmodellen mit FCM nach [Bal98]

In dieser Hierarchie stehen auf Blattebene die *Metriken*. Der Ansatz Goal Question Metric (GQM) beschreibt ein Verfahren zur Erstellung von Qualitätsmodellen, der sich gut mit FCM kombinieren lässt. Auf diese Weise lassen sich auf einfache Art und Weise Eigenschaften finden, die eine Dekomposition erlauben und schließlich Metriken anbieten. Bei GQM werden

Auswertungsziele für das zu erstellende Modell formuliert. Mit Hilfe von Metriken können dazu passende Fragen beantwortet werden. Der Nachteil dieser beiden Ansätze zur strukturierten Erstellung von Qualitätsmodellen ist, dass nicht klar zwischen Eigenschaften des Systems bzw. Produktes und Aktivitäten, die im System ausgeführt werden, getrennt werden kann. Zudem existiert keine Möglichkeit der Modellierung von Beziehungen zwischen Eigenschaften.

Vorteile in dieser Hinsicht bietet die Modellierung nach Activity-Based Quality Model (ABQM) [Loc10]. Dort wird zu einer Aktivität – beispielsweise *Wartung* – ein Fakt angegeben, der sich aus einer Komponente des Systems und einer Eigenschaft dieser zusammensetzt, beispielsweise *Quellcode* und dessen *Strukturiertheit*. Der Einfluss, den der Fakt auf die Aktivität hat, kann entweder positiv oder negativ sein. Dieser Zusammenhang ist auf Metamodellebene auch aus Abbildung 3.3 ersichtlich. Konkret im zuvor genannten Beispiel würde sich als ABQM-Ausdruck `[Quellcode|Strukturiertheit]` ⟶ `+[Wartung]` ergeben. Die Strukturiertheit des Quellcodes hat hier einen positiven Einfluss auf die Wartung des betrachteten Systems. Somit ist eine Modellierung auf feinerer Granularität als über FCM und GQM möglich.

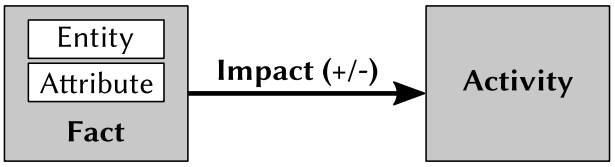

Abbildung 3.3: Metamodell zu ABQM nach [Loc10]

Ontologiesprachen sind eine geeignete Form der Wissensrepräsentation, die auch sehr mächtige Mittel zur Erstellung von Qualitätsmodellen bieten. Neben dieser Flexibilität versprechen sie eine gute maschinelle Verarbeitbarkeit durch einschlägige Frameworks. Mit Resource Description Framework (RDF) und Web Ontology Language (OWL) ist die feingranulare Eigenschaftsmodellierung mit der Abbildung von Beziehungen zwischen Eigenschaften ohne Weiteres erreichbar. Somit sind sie auch für die Modellierung von Qualitätseigenschaften in Web-Mashups geeignet. Jedoch ist die Anpassung an den Verwendungszweck – hier eine Vormodellierung von Wissen bzw. Regeln über die geplante Struktur des Eigenschaftsmodells – notwendig. Insgesamt ist eine Kombination der Aspekte der FCM-Methode mit den Vorteilen und Rahmenbedingungen einer Mehrzwecksprache zur Wissensrepräsentation empfehlenswert, um eine maximale Flexibilität gewünschter Zusatzfunktionalität und Facettierung zu erreichen. Aus FCM sollte dabei die Möglichkeit der übersichtlichen Aufteilung und Hierarchisierung in Teileigenschaften übernommen werden. Die Referenzierung von Bestandteilen, Bereichen bzw. Komponenten des Systems – hier insbesondere: der kompositen Anwendung – kann nach dem Vorbild von ABQM erfolgen. Der daraus entstehende *Fakt* wird anschließend mit einer Metrik versehen. Die charakteristischen Eigenschaften für den gewünschten Anwendungstyp Web-Mashup können somit unter Kombination dieser Strukturierungstechniken in einer gut integrierbaren Allzwecksprache wie OWL automatisiert verarbeitbar umgesetzt werden. Schließlich ist ebenfalls zu bedenken, wie gut sich die hinzukommenden Modelle für Qualitätseigenschaften in die bestehende Modellierung der Zielinfrastruktur integrieren lassen. CRUISE als Mashup-Plattform ist in seinen wichtigsten Infrastrukturkomponenten, der MRE und dem CoRe, auf die Verarbeitung von RDF-Daten ausgelegt und lässt somit eine nahtlose Integration zu.

3.3 Anforderungsmodellierung mit Aufgaben und Fuzzy-Mengen

Als zunächst vom Typ des entstehenden oder zu verändernden Anwendungssystems unabhängig kann die Aufgabenmodellierung (englisch: *task modeling*) als beliebte Technik der Formulierung und Dekomposition von Anforderungen betrachtet werden. Speziell für komposite Web-Mashups im Kontext von Geschäftsprozessmodellen wurde in [Tie+13] ein aufgabenorientiertes Anforderungsmetamodell vorgestellt. Es bietet Dekomposition von Aufgaben basierend auf einem Vokabular von Aktionen an, die funktionale Fähigkeiten repräsentieren. Qualitätsanforderungen werden hierbei allerdings nicht berücksichtigt. Haupteinsatzzweck dieser Aufgabenbäume ist vor allem die Strukturierung von unternehmensrelevanten Vorgängen, ähnlich wie es bei der Modellierung von Geschäftsprozessen der Fall ist. Besonders aufgrund ihres ebenfalls dekomponierbaren Charakters sind sie von ihrer Struktur und Zielstellung jedoch analog zu kompositen Mashups anzusehen. Die Modellierungskonzepte folgen den Paradigmen klassischer Aufgabeneditoren zur Unterstützung der Prozesse des Requirements-Engineerings. Neben verschiedenen schon länger existierenden Werkzeugen zur Beschreibung und Dekomposition solcher Aufgabenbäume wie *CTTE* [MPS02] stellt [Tie15, Kapitel 7] mit dem *DEMISA Task Model Editor* ein Werkzeug zur semantischen Aufgabenmodellierung vor. In erster Linie werden semantische Modelle, die innerhalb der Schnittstellenbeschreibungen von Mashup-Komponenten beispielsweise als *Capabilitys* genutzt werden, ebenfalls zur Dekomposition der Aufgabenbäume herangezogen, um so ein besseres Matching über Empfehlungsalgorithmen zu erreichen.

Bei der Spezifikation von Qualitätsanforderungen kann zwischen impliziten Techniken, wie Kontextmonitoring oder Tracking von Nutzerprofilen, und expliziten Techniken, die spezifische Interaktionspunkte über ein Frontend anbieten, unterschieden werden. Explizite Techniken, wie Faceted Browsing bzw. Weighted Faceted Browsing (WFB), vgl. [Pol09] und [Voi15, Abschnitt 3.3], bestehen aus einer Liste verfügbarer Filter, einem Modul zum Sortieren und Gruppieren der Ergebnismenge sowie einer Repräsentation dieser Ergebnisse, die iterativ verfeinert werden können. Einfache Formen des WFB findet man in Produktsuchportalen, die jedoch oftmals keine Wichtung der ausgewählten Eigenschaften anbieten. Dabei sind diese Ansätze auf bestimmte Anwendungsdomänen bzw. Produkttypen zugeschnitten. Für Qualitätseigenschaften in Web-Mashups existieren bisher keine umfassenden Ansätze, die sich das WFB zunutze machen.

Um einen einfachen Zugang mit unscharfen Ausdrücken in anpassbaren Qualitätsanforderungen zu ermöglichen, siehe Szenario ①, sind *Fuzzy-Mengen* (englisch: *fuzzy sets*) als Basistechnologie geeignet. Explizite Suchinterfaces, die Fuzzy-Anforderungen unterstützen, sind *Ask Fuzzy* [KKA12], *Fuzzy Query Interface* [RM03] und *Fuzzy Slider* [Ter+12]. Bei *Ask Fuzzy: Attractive Visual Query Builder* wird ein Assistent zur Formulierung von Datenbank-Anfragen vorgestellt, siehe Abbildung 3.4. [CL07] schlägt ein Fuzzy-Empfehlungssystem für Consumer vor. Ziel ist in jedem Fall die Vergrößerung der Zielgruppe an Benutzern für Auswahlprozesse der Mashup-Komponenten durch die Kombination der Spezifikation solcher multidimensionalen Anforderungen mit einem hohen Abstraktionsniveau. Hauptnutzungsmöglichkeit für derlei Prozesse sind Empfehlungssysteme, die sich die automatische Bewertung zunutze machen. Zusammen mit existierenden Rating-Ansätzen, etwa aus dem Web-Service-Bereich, vgl. [TT08], besteht so die Möglichkeit einer kombinierten Bewertungsvorschrift, die sich auf Konzepte der Fuzzy-Sets und Fuzzy-Logik (englisch: *fuzzy logic*) in den zugrunde liegenden Modellen und

den Entwicklungswerkzeugen manifestiert. Die detaillierte Vorstellung aller benötigten theoretischen Grundlagen, die für die Spezifikation und Auswertung unter Nutzung der Fuzzy-Sets benötigt werden, erfolgt in Abschnitt 4.4.

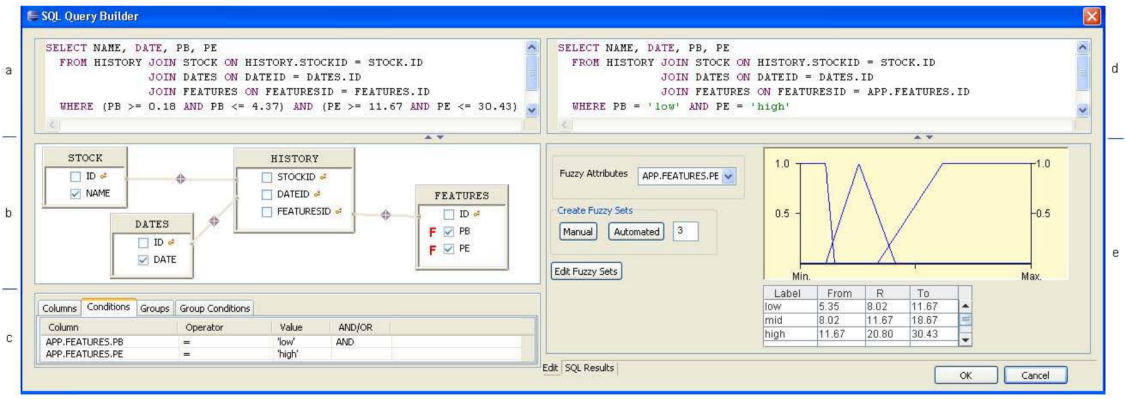

Abbildung 3.4: User Interface in Ask Fuzzy [KKA12] – auf der rechten Seite oben die Verwendung unscharfer Terme (*low* und *high*), darunter die Visualisierung der Membershipfunktionen

Implizite Techniken der Anforderungsüberwachung und -auswertung konzentrieren sich vor allem auf die fortschreitende Automatisierung einzelner Schritte im Gesamtprozess der Spezifikation. Hauptziel ist hierbei die Reduktion der vom Nutzer wahrgenommenen Ablenkung von seiner tatsächlichen Aufgabe bzw. des Aufwandes der bei der expliziten Spezifikation entsteht. Existierende Arbeiten setzen dazu ein Protokoll (englisch: *user log*) ein, das automatisch erstellt wird und als Grundlage für Inferenzoperationen dient, die Anforderungen im Hintergrund generieren bzw. ohne explizite Nutzerinteraktion ableiten. Eine Anpassung – beispielsweise die Konfiguration von Skalen – kann durch den Nutzerkontext erfolgen [HM95]. Eine Möglichkeit zur nutzerdefinierten Konfiguration sind *reference fuzzy sets*. Ein großes Problem dieser impliziten Methoden ist die ungewollte Ableitung bzw. versehentliche Angabe von Informationen, die der Nutzer über eine explizite Technik nicht in dieser Form vorgenommen hätte. Außerdem muss eine Mindestmenge an initialen Daten gesammelt werden, um aussagekräftige Anforderungen zu erzeugen (*sparsity*). In der stabilen Phase hingegen werden sich über die Zeit ändernde Nutzeranforderungen möglicherweise nicht angemessen behandelt. Ein einfacher Ansatz zur Extraktion von *fuzzy rules* mit einer Möglichkeit zur Behandlung der Anforderungsevolution – als Technik zur iterativen Verbesserung von Empfehlungen in Verbindung mit *neural learning* – wird in [CFT07] vorgestellt.

Noch haben die bei kommerziellen Produktsuchmaschinen beliebten Techniken der facettierten Suche – und damit der facettierten Anforderungsgewinnung – keinen Einzug in die Entwicklungsprozesse kompositer Web-Mashups erhalten. Zusammen mit dem Einsatz von Fuzzy-Sets ergibt sich somit eine aussichtsreiche Kombinationsmöglichkeit der Unterstützung von Qualitätsanforderungen in Web-Mashups mit den bestehenden Methoden etablierter Facettensuchmaschinen, wie sie aktuell schon bei Preisvergleichsportalen, beispielsweise `amazon.de` oder `idealo.de` eingesetzt werden.

3.4 Bewertungskriterien und Übersicht der Cluster für existierende Arbeiten

Nach der Betrachtung wichtiger Standards zu Qualitätsmodellen, Ansätzen zur Strukturierung dieser Modelle und existierender Arbeiten zur Definition von Anforderungen mittels Aufgabenbäumen und Fuzzy-Sets soll nun ein Vergleich repräsentativer Forschungsarbeiten innerhalb der Web-Mashups und verschiedener Typen verwandter Anwendungssystemtypen gezogen werden. Die Vergleichbarkeit der vorgestellten Ansätze wird durch die Bewertung innerhalb verschiedener Kriterien zur Modellierung und Ausführung erreicht. So kann ein kompakter Überblick gewonnen werden, der die bessere Einschätzung individueller Ausrichtungen erlaubt. Dieser Abschnitt stellt die Bewertungskriterien detailliert mit Erläuterungen zu möglichen Ausprägungen vor, die in einer Farbskala visualisiert werden. Die detaillierten Ausführungen zu jedem *Cluster*, das jeweils repräsentative Arbeiten nach ihrem Architekturtyp für die Analyse gruppiert, folgen ab Abschnitt 3.5.

3.4.1 Übersicht der Kriterien und Skala der Bewertung

Die Untersuchung existierender Arbeiten bezieht sich auf folgende Kriterien. Abbildung 3.5 zeigt eine Übersicht der möglichen Wertausprägungen, die von *gut* bis *schlecht* reicht. Bewertet wird jeweils die generelle Eignung für die beschriebenen Anwendungsfälle im Kontext der betrachteten kompositen Web-Mashups. Zu jeder Kategorie wird anschließend näher erläutert, welche Aspekte bei der Bewertung berücksichtigt werden. Außerdem stellen exemplarische Ausprägungen mögliche Bewertungen vor. Dabei wird jeweils eine besonders schlechte, eine neutrale und eine besonders gute Variante beispielhaft angegeben.

1. Modellierung und Eignung von Qualitäts*eigenschaften* (PRM)
2. Modellierung und Formalisierung von Qualitäts*anforderungen* (RQM)
3. Integration in *Werkzeuge* und *Entwicklungsprozesse* (TDP)
4. Automatisierte *Auswertung* von Qualitätsanforderungen (EVA)
5. Adaption und Verwertung in der *Plattform* (APF)

Abbildung 3.5: Bewertungsskala für existierende Ansätze

3.4.2 Modellierung und Eignung von Qualitätseigenschaften

Grundlegend soll bewertet werden, wie insbesondere Qualitätseigenschaften des Softwaresystems innerhalb der Ansätze modelliert werden. Neben strukturellen Charakteristika und der Möglichkeit zur automatischen Verarbeitung, beispielsweise beim Schreiben von Kontextinformationen beim Auswerten von Qualitätsanforderungen, wird hier auch die Eignung der innerhalb des Ansatzes vorgestellten Eigenschaften für den Anwendungstyp Web-Mashup bewertet.

Vor allem werden folgende Fragestellungen bei der Bewertung berücksichtigt: Betrachtet das Modell Qualitätseigenschaften, die die visuelle Präsentation (UI) von Inhalten im Sinne einer durch den Nutzer bedienbaren Anwendung abdecken? Trifft das Modell zu den enthaltenen Eigenschaften Aussagen, wie einzelne Werte erfasst werden können, zu welchem Zeitpunkt und durch wen dies geschieht und existiert ein Vorgabewert, ein konkreter Datentyp und ein Wertebereich? Welchem Zweck werden die im Modell erfassten Qualitätseigenschaften zugeführt? Wie sieht der typische Verwendungsfall für die Modelldaten aus? Wo liegt der Fokus bzgl. Domäne und Anwendungstyp? Bewertungen für folgende exemplarische Ausprägungen der Modellierung für Qualitätseigenschaften sind:


Keine UI-spezifischen Elemente, keine Metriken


Elemente spezifisch für Web-Anwendungen


Eigenschaften für UI-Mashups, Extra-Relationen zwischen Elementen

3.4.3 Modellierung und Formalisierung von Qualitätsanforderungen

Die automatische Weiternutzung von Qualitätsanforderungen in verschiedenen Phasen der Anwendungsentwicklung und -nutzung erfordert auch eine Formalisierung dieser Anforderungen. Zunächst wird daher betrachtet, wie die untersuchten Ansätze eine explizite Spezifikation von Qualitätsanforderungen anbieten und welche Modellierungsgrundlage gewählt wurde oder ob lediglich implizite Anforderungen auf definierten Qualitätseigenschaften gelten. Sofern die Modellierung für Qualitätseigenschaften angeboten wird, steht außerdem im Fokus, inwieweit diese durch verschiedene an der Entwicklung und Nutzung der Anwendungssysteme beteiligte Rollen angepasst werden können. Ist die Lösung spezifisch für bestimmte Anwendungsdomänen oder -typen, so wird das Potenzial der Übertragbarkeit auf das in dieser Arbeit betrachtete Web-Mashup-Paradigma für beliebige Anwendungen eingeschätzt.


Keine oder nur implizite Qualitätsanforderungen


Vordefinierte Anforderungen spezifisch für Web-Anwendungen


Anpassbare Formalisierung für Qualitätsanforderungen

3.4.4 Integration in Werkzeuge und Entwicklungsprozesse

Während die Modellierung von Eigenschaften und Anforderungen grundlegende Voraussetzungen für eine hohe Automatisierung und somit die bestmögliche Unterstützung der Nutzer und Entwickler ist, nutzt erst die Integration in Entwicklungsprozesse und Werkzeuge, das volle Potenzial der angestrebten Vorteile aus. Dieses Kriterium bewertet zunächst, ob überhaupt Werkzeuge bzw. Entwicklungsprozesse betrachtet und beschrieben werden oder ein reiner Modellierungsansatz vorgestellt wird. Existiert ein Entwicklungsprozess bzw. eine Plattform zur Werkzeugunterstützung, so wird deren universelle Einsetzbarkeit bzw. Übertragbarkeit auf andere Anwendungsdomänen eingeschätzt. Im Sinne der integrierten Entwicklung und Nutzung

ist hierbei interessant, ob das Werkzeug oder der Prozess die Anreicherung von Anwendungen mit Qualitätsanforderungen durch Web-Nutzer ohne Programmiererfahrungen, evtl. sogar während der Entwicklung der Anwendung, vermag.


Standalone-Modellierung ohne Integration


Integration in domänenspezifischen Entwicklungsprozess


Umfassende Integration in Entwicklungsplattform

3.4.5 Automatisierbare Auswertung von Qualitätsanforderungen

Entscheidender Faktor zur Automatisierung des Mashup-Entwicklungsprozesses und zur Nutzung innerhalb der Regulierungsprozesse selbstadaptiver Anwendungen ist die Automatisierung der Anforderungsauswertung. Neben der grundsätzlichen Möglichkeit der maschinellen Verarbeitung werden die existierenden Ansätze außerdem darauf untersucht, in welchen Szenarien – zur Laufzeit und zur Entwicklungszeit – welche Form der Anforderungsauswertung unterstützt wird. Dieses Kriterium ist durch das Formalisierungserfordernis abhängig vom Kriterium der Modellierung von Qualitätsanforderungen.


Keine automatisierte Auswertung von Anforderungen


Auswertung zur Entwicklungszeit, manuelle Auswertung (ohne Automatisierung)


Auswertung zur Laufzeit

3.4.6 Adaption und Verwertung in der Plattform

Dieses Kriterium bewertet den letztlichen Nutzen der Modellierungs- und Ausführungsinfrastruktur für die beteiligten Rollen, indem die Umsetzung der qualitätsfördernden Konzepte in der Entwicklungs- und Laufzeitplattform eingeschätzt wird. Insoweit ist die Fragestellung bedeutsam, welche Möglichkeiten für Konsequenzen die untersuchten Ansätze bereithalten, die abhängig vom Ausgang der Auswertung von Qualitätsanforderungen eingeleitet werden können. Die Quantifizierung erfolgt durch die Arten von Aktionen, die jeweils unterstützt werden. Diese reichen vom einfachen Feedback des Auswertungsresultates an den Anwendungsnutzer über Empfehlungssysteme bis hin zur Selbstadaption der Anwendung in der Laufzeitplattform.


Keine Aktionen vorgesehen


Feedback und einfache Benachrichtigungen, Recommendation


Automatische Adaption der Plattform mit Menge von Aktionen

3.4.7 Clusterweiser Überblick repräsentativer Arbeiten

Die vorgestellten Bewertungskriterien werden auf existierende Arbeiten zu Qualitätsanforderungen und -eigenschaften sowie dazu passende Entwicklungsparadigmen und Werkzeugunter-

stützung angewendet, die sich bezüglich des betrachteten System- bzw. Anwendungstyps voneinander abgrenzen. Einzelne Papiere und Ansätze sind in Clustern zusammengefasst, um eine ganzheitliche Bewertung bestimmter Typen von Anwendungssystemen vornehmen zu können. Dabei wurde die in Abbildung 3.6 dargestellte Facettierung zugrunde gelegt. Eine Facette unterscheidet die Art der Bereitstellung und Integration von Anwendungs- bzw. Systembestandteilen in eine klassische bzw. unstrukturierte und eine dienstbasierte Vorgehensweise. Auf der anderen Seite wurde eine Unterscheidung in UI-lastige Anwendungen und Backendsysteme, wie sie im reinen Web-Service-Bereich häufig anzutreffen sind, vorgenommen. Aus dieser Facettierung entstehen vier Cluster, die die Basis der zusammenfassenden Bewertung darstellen, vgl. Abschnitt 3.5. Die wichtigsten Vertreter der Cluster sind exemplarisch:

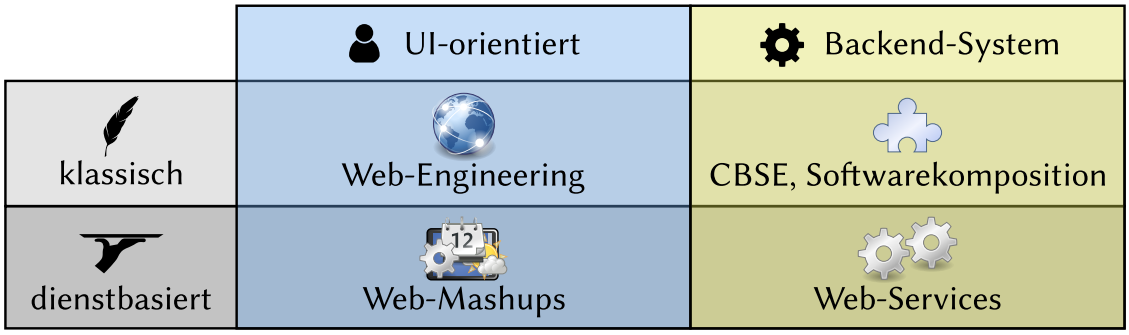

Abbildung 3.6: Cluster betrachteter Arbeiten und Ansätze

Web-Mashups: Ansätze zur Modellierung von Qualitätseigenschaften für Web-Mashups auf Ebene der Mashup-Komponenten [CDM09] und zur Aggregation auf Anwendungsebene [Cap+11a] sowie EUD-Kompositionsplattformen nach Cappiello u. a.

Web-Engineering: Modellierung von Qualitätseigenschaften für Anwendungssysteme, die in den klassischen Entwicklungsmodellen des Web-Engineerings entstehen, hier repräsentativ *New Generation Web Applications* [Ols+12] und das *2Q2U*-Framework [LOZ10] nach Lew, Olsina und Zhang

Web-Services: Modellierung von Qualitätseigenschaften und Vertragssprachen für Services ohne UI, Web Service Modeling Ontology (WSMO) [WSMO05], WS-QoS, WS-Policy [WS-Pol07] sowie abstrakte Service-Eigenschaften [OEH05] nach O'Sullivan u. a.

Softwarekomposition: Komposition und Beschreibung von Qualitätseigenschaften im Component-Based Software Engineering (CBSE), Verträge basierend auf nichtfunktionalen Anforderungen auf Code-Ebene [RZ07] nach Röttger und Zschaler

3.5 Qualität in Web-Mashups

Fixiert man existierende Arbeiten auf das Anwendungsparadigma Web-Mashups, so sind insbesondere die seit 2009 veröffentlichten Arbeiten zur Systematisierung von Qualitätseigenschaften von Cappiello, Daniel und Matera relevant für diese Arbeit. Initial wurde in [CDM09] ein

Modell für Qualitätseigenschaften solcher Web-Mashups vorgestellt. Der hierarchische Aufbau der Eigenschaften lehnt sich grob an den FCM-Ansatz und an die Vorgaben aus [ISO9126] an. Die generelle Dreiteilung in *API-Qualität*, *Datenqualität* und *Präsentationsqualität* ist in Abbildung 3.7 zu sehen. Ein Mehrwert soll durch das Modell sowohl für die Komponentenentwickler als auch für die Mashup-Entwickler entstehen, die diese Bausteine analog zu Beispielszenario ① aus dieser Arbeit zu Anwendungen komponieren. Das dort zugrunde gelegte Entwicklungsszenario unterstellt, dass beim Zusammensetzen des Mashups besonders *gute* Komponenten ausgewählt werden können. Ein Konzept der Spezifikation anpassbarer Qualitätsanforderungen fehlt stattdessen gänzlich, da die *Optimierungsbedingung implizit als Annahme* feststeht. Statt eines einheitlichen Komponentenmodells wird lediglich vorausgesetzt, dass die zu untersuchenden Komponenten ein oder mehrere APIs anbieten, die beispielsweise als Web-Service-Schnittstellen umgesetzt sein können.

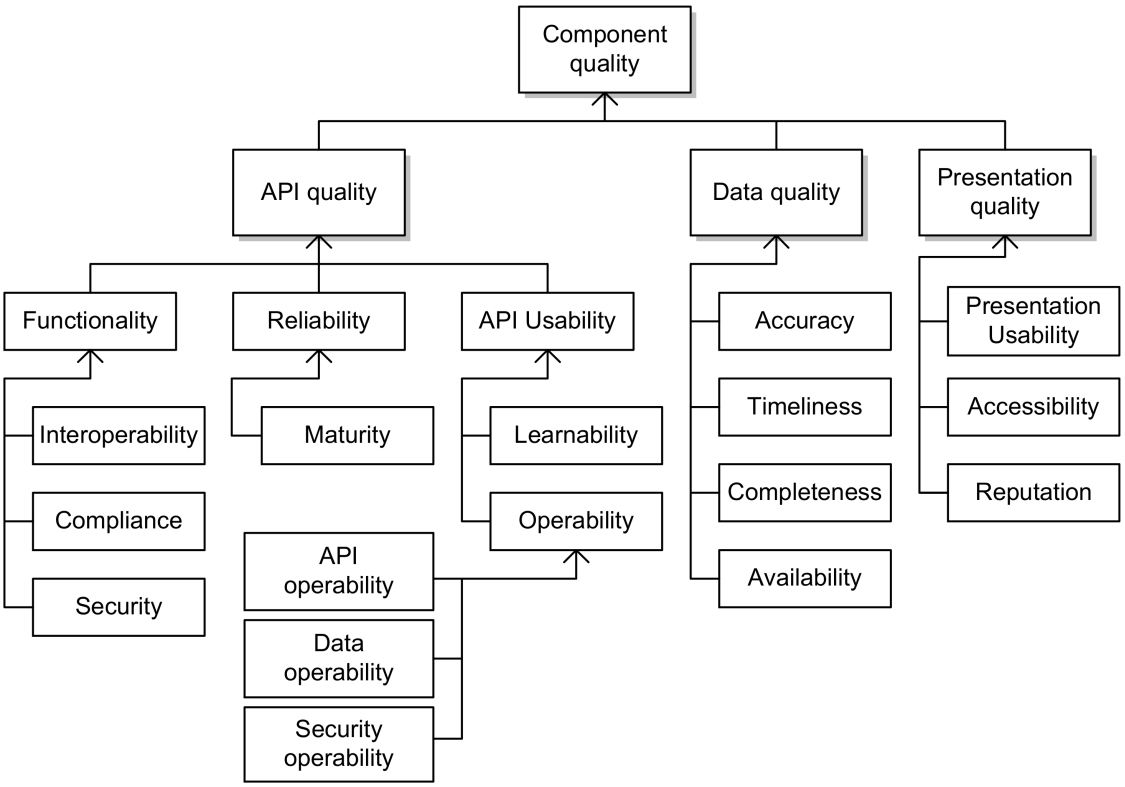

Abbildung 3.7: Qualitätsmodell für Mashup-Komponenten nach [CDM09]

Das vorgestellte Referenzmodell legt dabei großen Wert auf die API-Qualität und stellt Messmöglichkeiten für die einzelnen Eigenschaften vor. So wird beispielsweise die *Interoperabilität* einer Mashup-Komponente durch die Anzahl der angebotenen Protokolle, Sprachen und Datenformate bestimmt. Ähnlich verhält es sich mit der *Operability*-Berechnung bezüglich des API, der Daten und der Security. Innerhalb der Datenqualität wird die *Completeness* durch das Verhältnis aus der Menge der empfangenen Daten und der Menge der erwarteten Daten ermittelt. Das Modell ist ein erster wichtiger Ansatz zur spezifischen Betrachtung von Qualitätseigen-

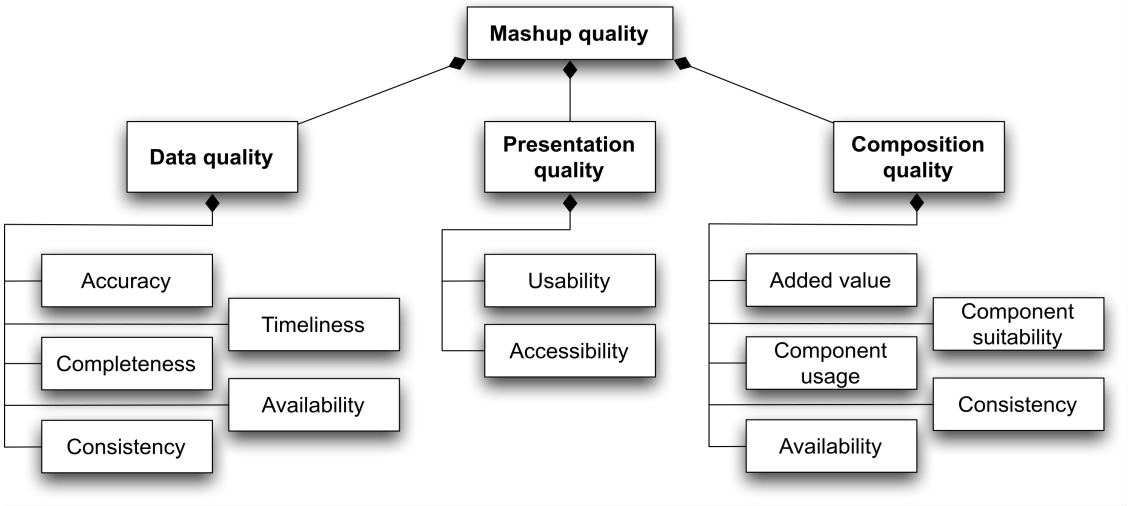

Abbildung 3.8: Qualitätsmodell für Mashups nach [Cap+11a]

schaften in Web-Mashups auf Komponentenebene. Die dafür aufgestellten Metriken zur Bewertung der einzelnen Größen kommen dabei neben Nutzerbewertungen im Usability-Bereich entweder mit sehr einfachen ganzzahligen Ausprägungen oder mit in der Praxis schlecht bestimmbaren und damit wenig aussagekräftigen Vorschriften aus.

Auf Anwendungsebene ergänzt [Cap+11a] neben den von Mashup-Komponenten projizierten Eigenschaften insbesondere die *Composition Quality* als Dimension von Qualitätseigenschaften das Qualitätsmodell für ganze Mashups. In Abbildung 3.8 sieht man im rechten Ast die daran beteiligten Eigenschaften. *Added Value* wird durch die Menge der angebotenen Daten und Features im Sinne der funktionalen Fähigkeiten bestimmt. Für *Component Suitability* und *Component Usage* existieren im Papier nur kurze Beschreibungen ohne Metrik. *Consistency* bezieht sich auf die syntaktische und semantische Kompatibilität der Kommunikationsschnittstellen der beteiligten Komponenten. Der Wert der *Availability* wird über die zeitliche Verfügbarkeit der Komponenten aggregiert. Mögliche Aggregationsmethoden werden hierbei nicht genannt.

Interessant dabei ist, dass einige der Eigenschaftswerte – vor allem die bereits aus dem Qualitätsmodell für Mashup-Komponenten bekannten – aus aggregierten Einzelwerten für Komponenten ermittelt werden können. Dabei werden allerdings höchstens nur Vorschläge für verschiedene generische Aggregationsfunktionen gegeben, ohne auf die Spezifika der Eigenschaftstypen einzugehen oder eine anpassbare Reihenfolge anzugeben. Für *Timeliness* kommen beispielsweise *Minimum*, *Average* und *Maximum* in Frage.

3.5.1 Ermittlung der Wertebelegung von Qualitätseigenschaften auf Anwendungsebene durch Aggregation

Der Gedanke der Aggregation von Komponenteneigenschaften, um Werte für die gesamte Komposition ableiten zu können, wird in [Pic+10] und [Cap+10] aufgegriffen. *Mashability* ist dort das Maß des Zusammenpassens eines neuen Bausteins zu schon verbauten Mashup-Komponenten innerhalb der Anwendung unter Zuhilfenahme der technologischen, syntaktischen und seman-

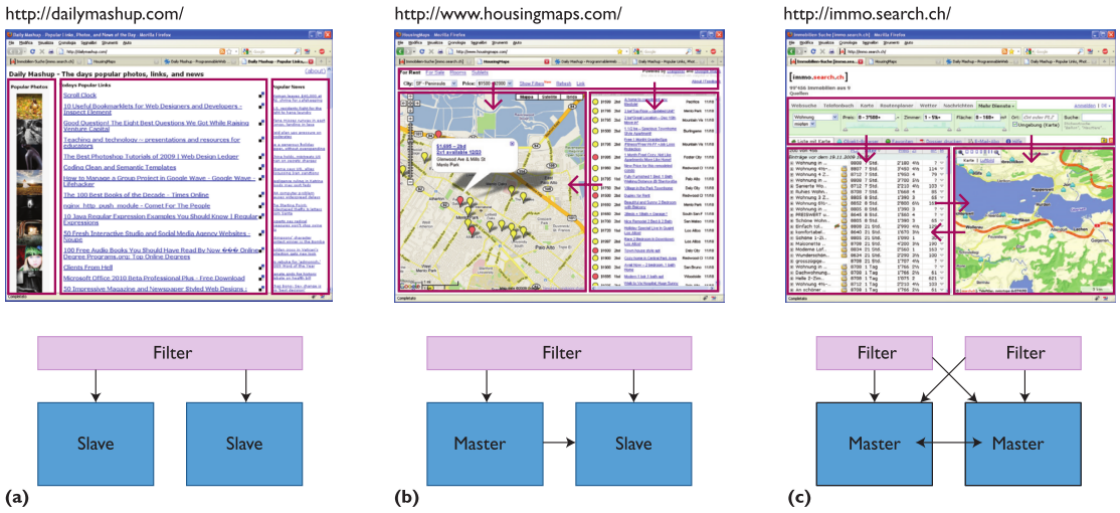

Abbildung 3.9: Kompositionsmuster für Mashups aus [Cap+10]

tischen Kompatibilität. Somit wird die potenzielle Verbesserung der Anwendung durch das Hinzufügen dieser Komponente abgeschätzt. Die tatsächlich aus unabhängigen Einzelwerten aggregierbaren Eigenschaften sind Accuracy, Completeness, Timeliness, Availability und Consistency. Der Gesamtwert wird dabei über eine Funktion ermittelt, die die jeweilige *Rolle jeder Komponente* innerhalb der Komposition berücksichtigt.

Zu diesem Zweck werden *Kompositionsmuster* vorgestellt, die die drei Rollen *Master*, *Slave* und *Filter* beinhalten können. Abbildung 3.9 zeigt drei mögliche Kompositionsmuster. Im mittleren Teil (b) der Abbildung wird beispielsweise die Kartenkomponente (Slave) der Anwendung housingmaps.com durch die Auswahl von Listeneinträgen gesteuert. Die obere Leiste dient als Filterkomponente zur Vorauswahl der Suchmenge. Bezüglich der aggregierten Werte gehen die Einzelwerte in einer unterschiedlichen Wichtung ein. Die *Accuracy* im Master-Slave-Pattern kann nun beispielsweise ermittelt werden, indem die Fehlerwahrscheinlichkeiten p der Einzelkomponenten addiert und von 1 abgezogen werden. Dabei wird die Fehlerwahrscheinlichkeit der Slave-Komponente s unter die Bedingung gestellt, dass die vom Master m gelieferten Daten korrekt sind: $\mathrm{Acc} = 1 - (p(e)_m + p(e_s|\,\mathrm{corr}_m))$.

Mit der Aggregation für bestimmte Qualitätseigenschaften beschäftigt sich ebenfalls das Papier »Modeling context-aware and socially-enriched mashups« [Tre+10]. Dort werden folgenden *Quality of Service Attributes* konkrete *Aggregation Patterns*, die als Funktionen angegeben sind, zugeordnet. Damit erfolgt im Gegensatz zum zuvor diskutierten Ansatz der Vorschlag von Referenzfunktionen für zumindest einige typische Eigenschaften im Anwendungstyp Web-Mashup. MC steht dabei für *Mashup Component*, Π für die Produktfunktion und min für die Minimumsfunktion:

Durchsatz: $\min(\mathrm{throughput}_{\mathrm{MC}})$ über die Anzahl der Anfragen, die in einem bestimmten Zeitintervall pro Komponente verarbeitet werden können

Verfügbarkeit: $\Pi(\mathrm{availability}_{\mathrm{MC}})$

Datenqualität: $\min(\mathrm{dq}_{\mathrm{MC}})$ ohne konkrete Spezifizierung der Einzelmetrik

Vertrauen: $\Pi(\mathrm{trust}_{\mathrm{MC}})$ mit Unterscheidung zwischen Human-Provided und Software-Based Services als grobe Entsprechung der Mashup-Komponenten mit und ohne UI

Die von Cappiello und Picozzi präsentierten Qualitätsmodelle stellen Eigenschaften und Optimierungsgrößen unabhängig von der Betrachtung der Anforderungen verschiedener im Prozess der Entwicklung und Nutzung beteiligter Rollen vor. Die Berechnung aggregierter Eigenschaftswerte bezieht die vorgestellten Kompositionsmuster unter Rollenzuweisung der Anwendungsbestandteile ein. Es wird jedoch ohne praktisch relevante Beispiele für die Wertermittlung argumentiert. Ebenso bleibt unklar, welche initialen Größen und Basiswerte durch vorherige oder statistische Bestimmung mit welchem Aufwand ermittelt werden müssen.

3.5.2 Empfehlungssysteme und EUD bei Mashups

Neben den reinen Modellierungsansätzen existieren bereits erste Plattformen, die grundlegende Kompositionsaufgaben in Mashups auch für Nutzer anbieten, die keine ausgeprägten Kenntnisse der Programmierung bzw. Modellierung besitzen. Assistenzfunktionen und Empfehlungsalgorithmen sollen die Eignung für sogenanntes EUD gewährleisten. Das Qualitätsmodell nach Cappiello u. a. wird innerhalb von *DashMash* – einer EUD-Plattform für Mashups – genutzt, die in [Cap+11c] und [Cap+11b] beschrieben wird. Das primäre Nutzungsszenario ist dort das Empfehlen von Mashup-Komponenten während der Anwendungsentwicklung. Hierbei wird die *Mashability*-Metrik – vgl. oben – verwendet, um das Zusammenpassen weiterer Komponenten zu einer bereits begonnenen Komposition zu bewerten. Neben der *Component Compatibility* fließt hier die *Aggregated Quality* ein, die sich aus gewichteten Einzelwerten zusammensetzt. DashMash versucht Nutzern ohne Programmierkenntnisse einen Zugang zur Komposition von Mashups durch einen leichtgewichtigen Entwicklungsprozess zu schaffen, indem für die Kernaufgaben eine angemessene Werkzeugunterstützung angeboten wird.

PEUDOM als *Platform for End User Development of Mashups* beschreibt in [Cap+12] als Erweiterung insbesondere einen empfehlungsbasierten Kompositionsworkflow, indem Alternativkomponenten angeboten werden, die die modellierten Qualitätskriterien besser erfüllen als ggf. bereits integrierte Anwendungsbestandteile. Die implizite Anforderungsgewinnung zielt dabei vor allem meist auf funktionale Eigenschaften, die etwa die Kompatibilität der Kommunikationsschnittstellen hinzuzufügender Komponenten bewerten. Dabei ist die Anforderungsquelle das bereits bestehende Anwendungsgerüst. Der Grobaufbau des Empfehlungssystems ist in Abbildung 3.10 zu sehen. Bewertungen der Qualitätseigenschaften fließen dort als *Quality Vector* in die Gesamtbewertung der Recommendation ein. Ergänzend zu PEUDOM wird in [Mat+13] die kollaborative Erstellung von Mashups mit dieser Plattform als Grundlage beschrieben. Kollaborative EUD-Mashup-Entwicklung liegt allerdings nicht im Fokus dieser Arbeit. *EnglishMash* bzw. *NaturalMash* [AP12] bietet zudem eine grundlegende Extraktion von Nutzerintentionen aus natürlicher Sprache in Textform mittels Natural Language Processing (NLP) an, die den EUD-Prozess der Komposition von Mashups mit eher explizit gewonnenen Anforderungen ergänzt.

Ein Ansatz, der wie die Arbeiten von Cappiello u. a. die Sammlung *ProgrammableWeb* [Pro11] als Vergleichsbasis nutzt und dort aus vorhandenen Mashups Empfehlungen für deren Bausteine generiert, ist »Data Source Recommendation for Building Mashup Applications« [CX10]. Dabei

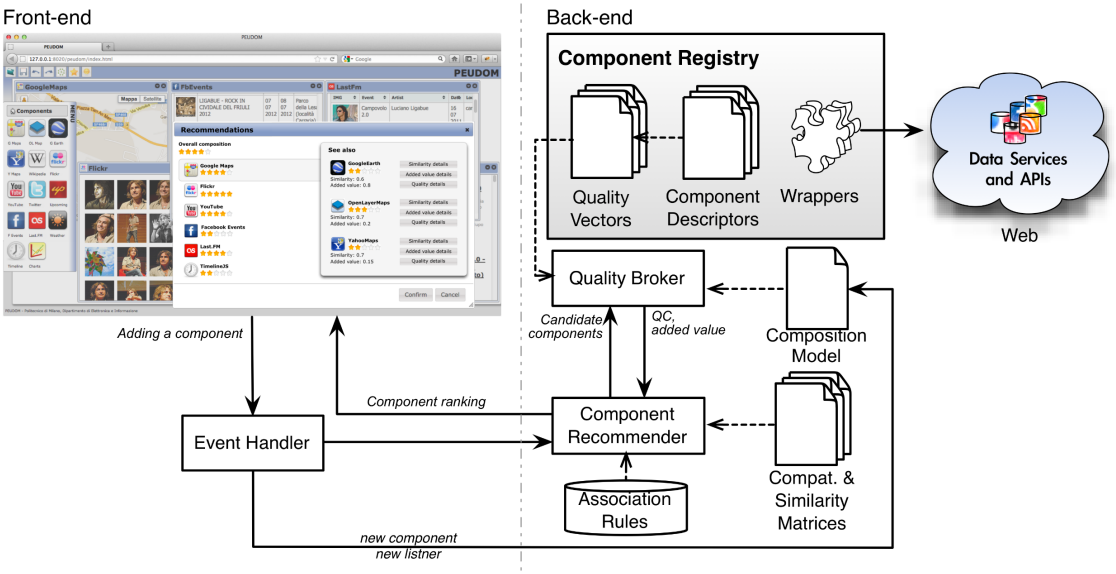

Abbildung 3.10: Architektur des Empfehlungssystems in PEUDOM nach [Cap+12]

wird ein Ähnlichkeitsmaß zwischen Komponenten basierend auf ihrer gleichzeitigen Verwendung in Mashups errechnet. Speziell für den Prozess des Austauschs von Komponenten innerhalb einer bestehenden Komposition – als Schritt der angestrebten Selbstadaptionsfähigkeit von Web-Mashups – ist dieser Anwendungsfall und das mit der Ähnlichkeitsermittlung verbundene Vorgehen interessant. Ein Werkzeug zur Empfehlung für die Mashup-Entwicklung wird in [Elm+08] mit »Mashup Advisor: A Recommendation Tool for Mashup Development« vorgestellt. Die Verkürzung der Zeit, die für die Entwicklung unter Nutzung von Assistenzfunktionen benötigt wird, ist dabei das Hauptziel. Mashup Advisor arbeitet mit einer bewerteten Liste von *Recommended Outputs* – einer Repräsentation einzelner gewünschter Ausgabeziele – zur Ergänzung eines in der Entwicklung befindlichen Mashups. Das Werkzeug errechnet daraus einen *Plan*, wie man mit den im Repository verfügbaren Services diese Zielvorgabe am besten erreichen kann und präsentiert die optimale Variante. Für die Berechnung werden Verwendungshäufigkeiten dieser Services in vorhandenen Mashups im Repository genutzt. Die Passgenauigkeit wird mit Hilfe eines *Semantic Matchers* berechnet, der ein Ähnlichkeitsmaß über die Anzahl übereinstimmender, textbasierter Tags unter Berücksichtigung von Synonymen nutzt. Mashup Advisor unterstützt damit die in dieser Arbeit geforderte Erweiterung vorhandener Anwendungen, die eine Grundlage für das vorgestellte Szenario der Live-Sophistication ist. Dabei unterstellt das Empfehlungswerkzeug ein vorhandenes Service- und Mashup-Repository mit annotierten Schnittstellen, um die Bewertung durchzuführen. Eine integrierte Weiterverwendung bzw. Automatisierung auf UI-Ebene ist aufgrund eines fehlenden Komponentenmodells jedoch erschwert.

Zum Konzept der Empfehlung innerhalb eines von Endnutzern getriebenen Mashup-Kompositionsprozesses gibt es auch innerhalb der dieser Arbeit zugrunde liegenden Mashup-Technologie Composition of Rich User Interface Services for Everybody (CRUISE) erste Ansätze. So wird in [Rad+12] ein hybrides Empfehlungssystem für Mashup-Komponenten – bzw. verallge-

meinert für Kompositionsfragmente – vorgestellt, das zur Grundlage Texteingaben mit zuvor in den Anwendungsbestandteilen annotierten *Capabilitys* abgleicht. Die Empfehlung der Kompositionsfragmente wird im Prozess der Anwendungserstellung und -erweiterung innerhalb der in [Rüm+11] skizzierten Plattform Engineering of Do-it-Yourself Rich Internet Applications (EDYRA) als EUD-Ergänzung von CRUISE genutzt. Ein semantisches Mediationskonzept erhöht dabei die Alltagstauglichkeit durch die automatisierte Auflösung von beispielsweise Synonymen oder bestimmten Begriffsbeziehungen, die den Einsatz unterschiedlicher semantischer Domänenmodelle erlauben, vgl. [Rad+14].

3.5.3 Fazit zum Stand der Forschung und Technik zur Mashup-Qualität

Der Artikel »Guest Editors' Introduction: Data Quality in the Internet Era« [BMS10] teilt die Grundmotivation und Rahmenbedingungen zur Nutzung und Entwicklung kompositer Web-Mashups dieser Arbeit: Jeder kann Daten und APIs bereitstellen und jeder kann sie nutzen, insbesondere um daraus Mashup-Anwendungen zu erstellen. Die Daten- bzw. Informationsqualität leidet bei der Durchführung dieser Prozesse potenziell, sodass als Herausforderung das *semantische Finden* und die *Passgenauigkeit* der erzielten Ergebnisse gemessen an der initialen Anwendungsidee bzw. den Anforderungen gelten. Die Schwäche bestehender Mashup-Plattformen ist, dass sie vor allem auf die *Erfassung und Verarbeitung funktionaler Anforderungen* begrenzt sind, sofern sie überhaupt Anforderungskonzepte unterstützen. Bedingungen über Qualitätseigenschaften werden höchstens *implizit* betrachtet, indem eine Optimierungsrichtung unterstellt wird. Beispielsweise wird pauschal angenommen, dass eine geringe Antwortzeit vom Anwendungsnutzer bzw. vom Entwickler gewünscht ist und somit zu einer besseren Bewertung führt. Vorherrschende Empfehlungssysteme orientieren sich an dieser Annahme, siehe [Pic+10]. Diese Pauschalisierung ist ein erheblicher Nachteil bestehender Forschungsansätze, da die anforderungsgetriebene Entwicklung erheblich eingeschränkt bzw. verhindert wird. Das Filtern und Suchen über Qualitätseigenschaften ist in aktuellen EUD-Plattformen zur Mashup-Entwicklung wie [Cap+11c], PEUDOM [Mat+13] und EnglishMash [AP12] allenfalls über eine eingeschränkte Menge an Eigenschaften möglich. In Summe stellen sie eine Basismenge an Referenzeigenschaften dar, die in dieser Arbeit in einem Modell für Qualitätseigenschaften genutzt werden kann, das speziell die Belange kompositer Web-Mashups abdeckt. Auch zur Eignung bestimmter Qualitätseigenschaften in Verbindung mit typischen Entwicklungs- und Nutzungsparadigmen von Web-Mashups werden in den untersuchten Arbeiten Aussagen getroffen, so z. B. in [Cap+10]: »Quality aspects such as maintainability, reliability, or scalability play a minor role because the final mashup is needed only for a short time.« Dies verdeutlicht die Heterogenität von Anforderungen am Beispiel der *Situational Applications*. Umfassende Konzepte zur Definition und Auswertung anpassbarer Qualitätsanforderungen, die in Empfehlungsprozesse einbezogen werden können, fehlen bisher vollständig in Ansätzen zu kompositen Web-Mashups. Dies macht deutlich, dass bestehende Forschungsansätze nur auf die *kurzfristige Nutzung spontan erstellter Mashups* ausgelegt sind. In dieser Arbeit wird der Lebenszyklus dahingehend erweitert, dass eine *iterative Anpassung von Anforderungen* erstrebt und durch die Entwicklungsprozesse und -werkzeuge umgesetzt wird.

Die Übersicht zu den Bewertungskriterien in Tabelle 3.1 zeigt daher für die Ansätze zur Qualität in Mashups, dass besonders die Modellierung von Qualitätseigenschaften sowie Empfehlungssysteme im Entwicklungsprozess umfassend – und zum in dieser Arbeit fokussierten An-

Tabelle 3.1: Bewertung der Arbeiten zur Qualität in Web-Mashups

PRM		Qualitätseigenschaften für UI, Metriken teilweise vorhanden
RQM		Keine Qualitätsanforderungen als Konzept
TDP		Entwicklungswerkzeug mit Unterstützung für EUD
EVA		Keine Auswertung von Qualitätsanforderungen
APF		Recommendation von Mashup-Komponenten

wendungstyp passend – untersucht wurden. Die Eigenschaftsmodelle lehnen sich an die Empfehlungen der ISO-250xx-Reihe an. Lediglich die teilweise wenig aussagekräftigen Metriken sind ein Defizit der Arbeiten zu diesem Anwendungstyp und erfordern hier zusätzliche Konzepte, um in einer automatisierten Umgebung nutzbar zu sein. Große Abstriche müssen hingegen bei der Anforderungsmodellierung und -auswertung gemacht werden. Der *Umgang mit Nutzerfeedback* wird im Stand der Forschung als noch immer offene Fragestellung angesehen. Die Erschließung von Qualitätsanforderungen über den Anwendungsnutzer als weitere Quelle ist daher als erwünschtes Forschungsergebnis gefordert. Werkzeugunterstützung für die Entwicklungsprozesse wird vor allem in den Arbeiten zur nutzergetriebenen Entwicklung grundsätzlich betrachtet. Sie erreicht jedoch nicht die Mächtigkeit bzw. Vielseitigkeit, die ein Entwicklungsprozess unter Nutzung der in dieser Arbeit initial definierten Quellen für Qualitätsanforderungen benötigt, vgl. Szenarien in Abschnitt 2.2.

3.6 Qualitätsanforderungen im Web-Engineering

Erweitert man den Rahmen der Betrachtung von spezifischen Paradigmen der Architektur und Entwicklung kompositer Web-Mashups auf Web-Anwendungen im Allgemeinen, so trifft man unter den existierenden Arbeiten zur Qualitätsmodellierung vor allem auf die Ansätze von Lew, Olsina und Zhang, vgl. unten. Grundsätzlich ohne Einschränkungen der Anwendungsdomäne werden bei ihnen insbesondere Modellierungs- und Evaluierungsmethoden für *Usability* und *User Experience* untersucht. Da generell im Web-Engineering – von der Strukturierung im feingranularen Document Object Model (DOM) abgesehen – als Träger für Qualitätseigenschaften keine modellhaft abgrenzbaren Komponenten analysiert werden können, stehen anwendungsweit geltende sowie die sich im UI bemerkbar machenden Charakteristika im Fokus.

Zur Auswertung von Qualitätseigenschaften für Web-2.0-Anwendungen wird in [LOZ10] das Framework *2Q2U* erstmals vorgestellt und in *Updating Quality Models for Evaluating New Generation Web Applications* [Ols+12] erweitert. 2Q2U steht dabei für die Hauptgegenstände der Untersuchung: *Quality*, *Quality in Use*, *Actual Usability* und *User Experience*. Bezüglich des Modells aus ISO 25010 werden zwei Charakteristiken zur Ergänzung vorgeschlagen: *Informationsqualität* und *Learnability in Use*. Zudem werden dort die Konzepte *Actual Usability* und *Actual User Experience* diskutiert. Der größte praktisch nutzbare Beitrag der Forschungsarbeit aus diesem Papier ist ein Prozess zur Auswertung und Verbesserung der *Quality in Use*. Abbildung 3.11

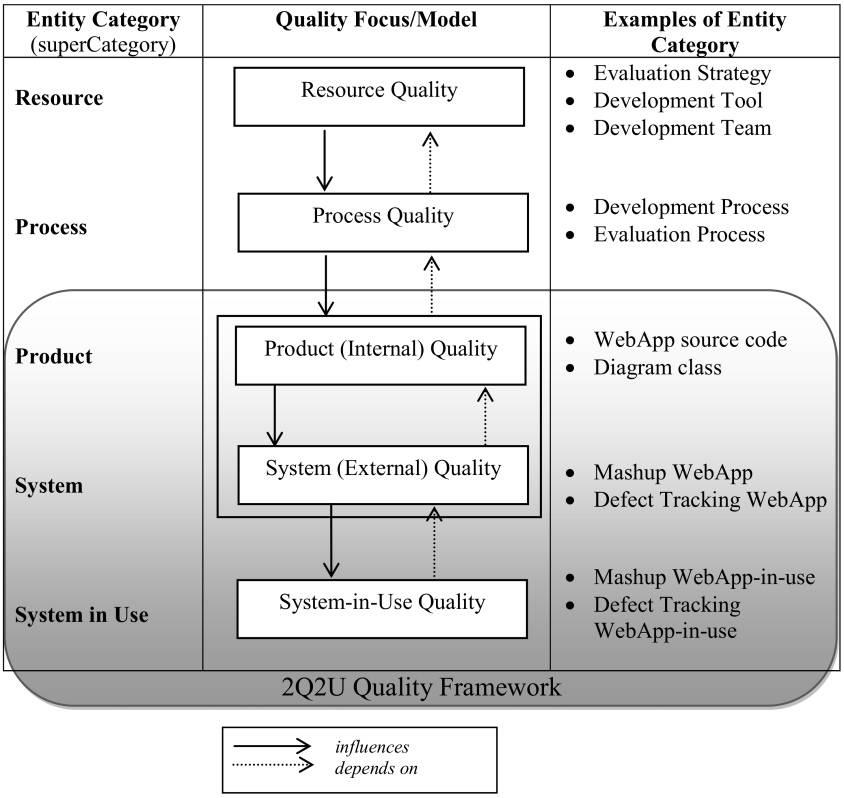

Abbildung 3.11: 2Q2U-Qualitätsframework aus [Ols+12]

zeigt innerhalb des vorgestellten Frameworks eine Abstufung der Qualitätsmodellierung abhängig von der betrachteten Granularität beginnend mit der Ressource über das Produkt bis zu einem in der Benutzung befindlichen System. Insbesondere der untere Teil der Qualität eines *System-in-Use*, welche stark von der Interaktion mit dem Anwender geprägt ist, stellt einen Hauptbeitrag des Frameworks dar. Web-Mashups treten zwar als Beispiele in der rechten Spalte auf, sind jedoch nicht als architektonisch besondere Web-Anwendungen, sondern nur als beliebige Anwendungsbeispiele zu verstehen.

Eigenschaften aus dem präsentierten Framework 2Q2U sind als Ergänzung des zu erstellenden Referenzmodells für Qualitätseigenschaften zu verstehen, insbesondere wenn die Qualität eines in der Benutzung befindlichen Systems bewertet werden soll. Allerdings erfolgt der Bewertungsprozess der Qualitätseigenschaften in den durchgeführten Beispielen der Evaluierung der Forschungsarbeiten komplett manuell. Der in dieser Arbeit angestrebte Auswertungsprozess für Qualitätsanforderungen zielt jedoch auf eine möglichst hohe Automatisierung ab. Allenfalls zuvor aggregierte Nutzerbewertungen, die zur Verwaltung in Repositorys geeignet sind, könnten als Pendant zur manuell gewonnenen Bewertung aus Fragebögen – wie es innerhalb der Auswertung nach dem 2Q2U-Framework vorgesehen ist – angesehen werden.

In »Modelling Web-Based Systems Requirements Using WRM« [MPT08] monieren Molina, Pardillo und Toval, dass ein hoher Anteil der Entwicklungsprozesse von Web-Anwendungen ohne das Management von Qualitätszielen arbeitet. Daher wird ein Anforderungsmetamodell

zur Unterstützung der ersten Phasen in Entwicklungsprozessen des Web-Engineerings vorgeschlagen. Als Werkzeugunterstützung wird ein Plugin für Entwicklungsumgebung Eclipse angeboten, das dieses Vorgehen praktisch unterstützt.

Tabelle 3.2: Bewertung der Arbeiten zu Qualitätsanforderungen im Web-Engineering

PRM		Qualitätseigenschaften für Web-Anwendungen
RQM		Qualitätsanforderungen nur konzeptionell vorhanden
TDP		Konzeptionelles Framework zur Auswertung
EVA		Manuelle Auswertung von Anforderungen zur Laufzeit
APF		Feedback an den Entwickler

Tabelle 3.2 fasst die Bewertungen zu den einzelnen Kriterien für die stellvertretenden Ansätze zur Qualitätsmodellierung und -auswertung im Web-Engineering zusammen. Qualitätseigenschaften werden dort für Web-Anwendungen vor allem im Bereich der User Experience betrachtet. Dies geschieht grundsätzlich *nur auf Anwendungsebene*, da die spezifische Ausrichtung auf Eigenschaften der Anwendungsbestandteile mit UI, wie sie als Black-Boxes in Mashups vorherrschen, fehlt. Für die Eigenschaftsmodellierung in dieser Arbeit können die untersuchten Usability-Ansätze aus dem Web-Engineering somit kaum einen Beitrag leisten. Wie bei den Arbeiten zu Web-Mashups ist hier eine Schwäche, dass *Qualitätsanforderungen nur konzeptionell vorhanden* sind. Über Frameworks werden zwar die Auswertungsprozesse grundsätzlich behandelt, jedoch ist die Auswertung manueller Natur im Sinne einer Anleitung. Dies ist vor allem darin begründet, dass ausschließlich fertige Anwendungen bewertet werden, die über einen klassischen Entwicklungsprozess entstanden sind. Eine situative Entwicklung und Assistenzfunktionen hierfür – wie etwa bei der Live-Sophistication – werden nicht berücksichtigt. Die generische Nutzbarkeit dieses Evaluierungsframeworks ist für diese Arbeit kaum gegeben, da eine automatisierte Verarbeitung zwingend erforderlich ist. Ansätze, die das Anforderungsmanagement für Web-Anwendungen – insbesondere mit Fokus auf Aufgabenbäumen als Variante der Anforderungsmodellierung – behandeln, wurden bereits in Abschnitt 3.3 betrachtet.

3.7 Qualitätseigenschaften und -anforderungen bei der Auswahl und Komposition von Web-Services

Web-Services als verteiltes System unterscheiden sich folgendermaßen von den bisher betrachteten Web-Anwendungen und im Speziellen von Web-Mashups. Komponierte Web-Services bilden keine Anwendungen, die zur Bedienung durch menschliche Benutzer vorgesehen sind, sondern es entstehen erneut Web-Services bzw. ausführbare Geschäftsprozesse, deren komplexe Funktionalität durch Clients genutzt werden kann. Unter den Kompositionstechniken bei Web-Services wird zwischen *Orchestrierung* – der Beschreibung des Aufrufs und der Abhängigkeiten von Web-Services in Geschäftsprozessen aus Sicht einzelner Services – und *Choreographie* – der Spezifikation der Fähigkeiten zur Interaktion von Web-Services, beispielsweise mit WS-CDL

[WSCDL05] – unterschieden. Daher entwickelte sich – lange bevor sich Technologien für Web-Mashups verbreiteten – das Bedürfnis nach einer Möglichkeit zur Angabe von Service Level Agreements (SLAs) zur Spezifikation der gewünschten Quality of Service (QoS) aus Sicht der Clients bei der Komposition von Web-Services. Da Web-Services grundsätzlich ohne UI existieren, sind typische Qualitätsanforderungen vor allem an den Ausführungsbedingungen der Geschäftslogik bzw. an der Güte der Netzwerkkommunikation orientiert. Eine Beispielanforderung an einen Web-Service könnte lauten, dass der Dienst die Antwort auf eine Anfrage immer innerhalb von zwei Sekunden geliefert haben soll.

Mit der Spezifizierung der Web Service Modeling Ontology (WSMO) wird in [WSMO05] versucht, möglichst umfassend die Charakteristika von Web-Services semantisch zu beschreiben. Dabei werden bestimmte nichtfunktionale Eigenschaften vorgeschlagen, die sich an die Dublin Core (DC) Metadata Initiative anlehnen, wobei die dort definierten klassischen Metadaten wie *owner*, *publisher*, *title* oder *version* keine Einschränkungen auf funktionalen Eigenschaften definieren, wie dies von Qualitätseigenschaften erwartet wird. In *Formal description of non-functional service properties* [OEH05] beschreiben O'Sullivan u. a. im Rahmen von WSMO die Modellierung abstrakter nichtfunktionaler Eigenschaften für Web-Services. Die Arbeit *Non-functional properties in Web services (D28.4 V0.2)* [Tom+08] fasst sowohl die Integration von nichtfunktionalen Eigenschaften aus [OEH05] in WSMO als auch die Bewertung zahlreicher älterer Ansätze der kompositen Softwarelandschaft – insbesondere im Bereich Web-Services – zusammen.

In »A unified description language for human to automated services« [Obe+13] streben Oberle u. a. die Konsolidierung von technischen, Nutzungs- und Geschäftsaspekten von Dienstleistungen mit Einführung der Unified Service Description Language (USDL) an. Besonders die Randbedingungen des geschäftlichen Einsatzes werden in Qualitätseigenschaften – wie Lizenzierung, Preismodelle und abstrakteren Service Levels – betrachtet. Das Ziel ist hierbei die Überbrückung der Grenzen zwischen vollautomatisierter und vom menschlichen Eingreifen bestimmter Welt. Auch Qualitätsanforderungen auf diesem Eigenschaftsmodell werden konzeptionell behandelt. Hierbei wird in Aussicht gestellt, sie durch Transformationen in ausführungsnähere Modelle überführen zu können.

Grundsätzlich bieten die Referenzmodelle der qualitätsgetriebenen Ansätze in Web-Services keine GUI-spezifischen Eigenschaften, da Dienste bzw. Anwendungen untereinander kommunizieren. Bei Web-Mashups erfolgt allerdings eine Kommunikation von Benutzern mit Anwendungen und Diensten, sodass die Berücksichtigung von UI-relevanten Qualitätseigenschaften fehlt, um sinnvoll auf den Anwendungstyp Web-Mashup projiziert zu werden. Dies gilt allerdings auch für alle anderen Ansätze, die sich mit Eigenschaften von Web-Services im engeren Sinne beschäftigen.

Zur Spezifikation nichtfunktionaler Anforderungen bei klassischen WSDL-basierten Web-Services eignet sich das bereits bei den Standards in Abschnitt 3.1 vorgestellte Framework *Web Services Policy* [WSPol07]. Jedoch wird durch den Standard nur die äußere Struktur im XML-Format vorgegeben. Die Möglichkeiten semantischer Interpretation und damit der Anforderungsauswertung bleiben offen, vgl. Abschnitt 3.1. Exemplarisch zeigt »A framework for QoS-based Web service contracting« [CP09], dass viele Ansätze zur Berücksichtigung von Web-Service-Anforderungen sich auf automatisiertes Aushandeln von Kontrakten konzentrieren. In dieser Arbeit soll die Anwendungsadaption allerdings primär durch nutzergetriebene Auswahlprozesse entstehen, die durch Empfehlungssysteme unterstützt wird. Ein Effekt, der sich aus

dieser abweichenden Haltung ergibt, ist, dass QoS-Vertragssprachen bei Semantic Web Services (SWS) generell sehr komplex und damit für Web-Nutzer, die am Adaptionsprozess ihrer komponierten Anwendungen beteiligt werden sollen, ungeeignet sind. Dies gilt insbesondere für die ersten beiden Referenzszenarien dieser Arbeit.

Die aus ISO/IEC 25010 bekannte *Quality in Use* ist auf Ausführungsebene auch bei Web-Services relevant. Hier ist zu berücksichtigen, welche Clients als *Nutzer* einzelner Dienste auftreten. Thoss verwendet im Rahmen ihrer Arbeit »Systemunterstützung zur Bewertung der Qualität persönlicher Cloud-Dienste« [Tho14] synonym den Begriff *Quality of Experience*. Dort wird abstrahiert auf Cloud-basiert angebotene Dienstleistungen untersucht, wie man als Nutzer einen qualitativen Überblick auf persönlich genutzte Cloud-Dienste gewinnen kann. Projiziert auf das anforderungsgetriebene Vorgehen in dieser Arbeit ist das Paradigma jedoch nur als Vorstufe der nutzerzentrierten Komposition von Mashups in Szenario ① nutzbar. Eine automatisierte Adaption basierend auf zur Laufzeit ermittelten Messwerten findet nicht statt.

Tabelle 3.3: Bewertung der Arbeiten zur Komposition von Web-Services

PRM	⇊	Keine Qualitätseigenschaften mit UI-Bezug
RQM	⇈	Unterstützung für anpassbare Anforderungen
TDP	↕	Umsetzungen jeweils domänenspezifisch
EVA	↑	Laufzeitauswertung von Anforderungen
APF	↕	Automatisierte Auswertung

Zusammenfassend zeigt Tabelle 3.3 die Vor- und Nachteile der Ansätze zur Qualitätsmodellierung und -auswertung bei Web-Services. Die Spezifikation von Kontrakten, vor allem zur automatischen Aushandlung von Qualitätsparametern ist in der Komposition von Web-Services bereits sehr ausgeprägt erforscht. Jedoch ist die Ausprägung sehr komplex und es mangelt an Implementierungen. Konkret heißt das, dass die Strukturen zur Formulierung von Anforderungen zwar existieren, die Semantik zur vollautomatischen Nutzung von Auswertungsergebnissen auf Messwerten für Qualitätseigenschaften grundsätzlich nicht definiert ist und somit nicht ohne Weiteres als Quelle für die Mashup-Plattform genutzt werden kann. Wie für das Paradigma der Web-Services üblich, werden zustandslose Komponenten bzw. Services mit Eigenschaftsmodellen beschrieben, die ohne UI-Bezug sind und dadurch die Interaktion mit menschlichen Benutzern nicht hinreichend abdecken. Die Web-Service-Eigenschaften, die auch typisch für den Einsatz in der Mashup-Plattform sind, werden in das Konzept dieser Arbeit einbezogen und sind beispielsweise im Referenzmodell für Qualitätseigenschaften vertreten, vgl. Abschnitt 4.6.

3.8 Qualitätsanforderungen in kompositen Softwaresystemen

Auch klassische Softwaresysteme nutzen Komponenten zum Aufbau komplexer Anwendungen. Das Component-Based Software Engineering (CBSE) beschäftigt sich dabei mit der Komposition von Softwarekomponenten, insbesondere mit Komponentenmodellen und Kompositions-

sprachen. Trotz der im Vergleich zu Web-Services starken Kopplung an bestimmte Sprachen und Technologien existieren in der Softwarekomposition auf Code-Ebene bzw. in der Granularität des Abstract Syntax Graph (ASG) Schnittstellenbeschreibungen, die eine geeignete Wiederverwendung und den Vergleich von Softwarekomponenten ermöglichen. Röttger und Zschaler stellen in [RZ07] bzw. [RZ03] in diesem Zusammenhang CQML$^+$ vor. Diese Vertragssprache erlaubt die Formulierung von Anforderungen für Qualitätseigenschaften auf Softwarekomponenten, vgl. Codebeispiele in Abbildung 3.12. Sie basiert dabei auf der Component Quality of Service Modeling Language (CQML), vgl. [Aag01], und erfährt mit der Energy Contract Language (ECL) eine Erweiterung insbesondere im Bereich der Anforderungen zur Energieoptimierung auf Softwareebene. Damit wird es beispielsweise möglich, die Energieauswirkungen von Zustandsübergängen bei Hard- und Softwarekomponenten so zu beschreiben, dass sie in Verbindung mit Nutzeranforderungen für Empfehlungsprozesse oder die Selbstadaption hilfreich sind. Bei den zugeordneten Adaptionsaktionen handelt es sich laut [Göt+12] um *domänenspezifische Rekonfigurationen*, wie etwa die Migration von Softwarekomponenten. CQML ermöglicht zudem die Aggregation von QoS-Eigenschaften. Das Konzept der Aggregation auf struktureller sowie auf Ebene der Wertebelegung spielt in dieser Arbeit ebenfalls eine Rolle. Es wird innerhalb der Beschreibung des Eigenschaftsmodells in Unterabschnitt 4.5.2 aufgegriffen. Als Nachteil sehen Röttger und Zschaler ebenfalls, dass nur Eigenschaften auf Ebene der Softwarekomponenten betrachtet werden und keine Anforderungen, beispielsweise über die Nutzung von Ressourcen, möglich sind. Außerdem können Abhängigkeiten zwischen verschiedenen Eigenschaften – etwa im Sinne von ABQM – nicht ohne Weiteres spezifiziert werden.

Graphical Model Element	CQML$^+$ Code
<<characteristic>> response_time value<500	`quality GeneratedStmtID (op: Operation) { response_time(op) < 500; }`
<<characteristic>> response_time value<500 [for op]	`profile GeneratedProfileID for ComponentName { provides GeneratedStmtID (op); }`

Abbildung 3.12: CQML$^+$-Ausschnitte aus [RZ07]

Auf UML-Ebene selbst bietet die Object Constraint Language (OCL) eine Verfeinerung der graphischen Modellierung von Softwaresystemen. Dabei erfolgt eine textuelle Beschreibung von *Constraints*, die als Bedingungen von Qualitätsanforderungen betrachtet werden können. Derar-

Tabelle 3.4: Bewertung der Arbeiten zu Anforderungen in kompositen Softwaresystemen

PRM		Granularität: Qualitätseigenschaften auf Low-Level-Code
RQM		Formalisierte Bedingungen der Anforderungen in CQML$^+$
TDP		Integration in Framework
EVA		Framework
APF		Nutzung zur Laufzeit

tige OCL-Zusicherungen, wie die Invariante auf einer Klasse – beispielsweise: `context Member inv: self.birthyear > 1890` – existieren als Spezifikationsform nur losgelöst von den in dieser Arbeit benötigten Angaben eines Ereignis-Triggers zur flexiblen Verwendung innerhalb der Referenzszenarien und der Zuordnung von Adaptivitätsaktionen. In Tabelle 3.4 werden die Charakteristika der Anforderungshandhabung in den ausgewählten Systemen zur Softwarekomposition bezüglich der Bewertungskriterien beschrieben.

3.9 Fazit zum Stand der Forschung und Technik

Zwischen den betrachteten Typen verschiedener Anwendungssysteme bestehen große Unterschiede sowohl bei den Modellierungsgrundlagen als auch beim Umgang mit Qualitätsanforderungen. In der Analyse wurde dabei besonders auf die konzeptionelle Erschließung und Umsetzung von Infrastrukturen zur Spezifizierung und Auswertung von Qualitätsanforderungen geachtet. Vor allem die Integration in Nutzungs- und Entwicklungsprozesse wird aus Sicht der Laufzeitinfrastruktur und Werkzeugunterstützung aufgrund der historischen Rollenzuordnung und der Entwicklungsziele in vielen Fällen vernachlässigt. Die Anforderungsevolution, d. h. die iterative Anpassung von Anforderungen, endet bei klassischer Software und den damit verbundenen Entwicklungsparadigmen – abgesehen von spezifischen Wartungsverträgen – typischerweise spätestens mit der Auslieferung an den Kunden. Durch die davon abweichende Rollenbelegung bei der individuellen Entwicklung von Web-Mashups ergibt sich eine *abweichende Motivation im Umgang mit Anforderungen*. Es besteht ein stärkeres Bedürfnis zum kontextabhängigen Mitführen von formalisierten Anforderungen, das in anderen Arbeiten zwar zum Teil festgestellt, jedoch weitgehend nur rudimentär gedeckt wird.

Tabelle 3.5 zeigt eine Übersicht bewerteter Ansätze auf Ebene der analysierten Cluster, siehe Abbildung 3.6. Dabei fällt zunächst auf, dass Modelle über Qualitätseigenschaften für Mashups und Web-Anwendungen in aktuellen Arbeiten vorgeschlagen wurden, jedoch *keine Berücksichtigung anpassbarer Anforderungen* aufweisen und durch *fehlende Metriken und Automatisierung* gekennzeichnet sind. Ziel dieser Arbeit ist es, diese zentrale Schwäche durch ein Konzept für das automatisierte Auswerten anpassbarer Qualitätsanforderungen für Web-Mashups zu beheben. Die vielfältigen Qualitätseigenschaften, die in den Mashup-Ansätzen vorgeschlagen wurden, werden im Referenzmodell dieser Arbeit aufgegriffen, vgl. Abschnitt 4.6. Vertragsmodelle und -sprachen, die ähnlich zu den in dieser Arbeit angestrebten Qualitätsanforderungen schon

seit Jahren im Bereich von Web-Services erforscht werden, haben große Unterschiede in der grundlegenden Eigenschaftsmodellierung. Die Eigenschaften, auf die Bezug genommen wird, entstammen den Schnittstellenbeschreibungen der Web-Services, die naturgemäß keine Charakteristika der Nutzerschnittstelle abdecken. Vor allem in der nutzergetriebenen Entwicklung von UI-Mashups haben diese jedoch einen hohen Stellenwert. Ansätze in der Anforderungsmodellierung semantischer Web-Services weisen außerdem nur unzureichende Implementierungen auf und sind bezüglich der Werkzeugunterstützung und der Ausrichtung ihrer Nutzergruppen umständlich in der Handhabung. Während Web-Services zur Eigenschaftsmodellierung somit kaum einen Beitrag zu den Konzepten dieser Arbeit leisten können, sind sie ein Vorbild für die anforderungsgetriebene Komposition und die grundlegende Anforderungsformalisierung. Ähnlich gut aufgestellt bei der Modellierung von Qualitätsanforderungen sind die Ansätze des CBSE, die vielversprechende Möglichkeiten zur Formulierung von Constraints bieten. Allerdings stützen sich diese Anforderungen auf quellcodenahe Eigenschaften bzw. ASGs, anstelle abstrahierte Charakteristika zu nutzen, die sie in der nutzergetriebenen Entwicklung benötigt werden. Der Beitrag dieses Clusters ist somit vornehmlich als Inspiration für die Art der Anforderungsmodellierung zu bewerten. Ein umfassendes Zuordnen von automatisiert ausführbaren Handlungsvorschriften zu Qualitätsanforderungen muss in dieser Arbeit allerdings neu konzipiert werden. Die Energy Contract Language (ECL) liefert dafür ein exemplarisches Vorgehen, ist jedoch mit den Rekonfigurationshinweisen auf die Energiedomäne beschränkt. Ziel dieser Arbeit ist der Aufbau einer umfassend nutzbaren Adaptionsplattform, die sich Bedingungen auf Qualitätseigenschaften verschiedener Quellen zunutze macht. Existierende Arbeiten über Qualitätseigenschaften bei klassischen Web-Anwendungen, die keine explizite Komponenten- oder Dienstarchitektur aufweisen, behandeln in der Hauptsache Usability-Kriterien. Da hier keine Empfehlungssysteme oder Auswahlprozesse eine Rolle spielen, ist vor allem die *Quality in Use* Gegenstand der Untersuchung. Im Referenzmodell für Qualitätseigenschaften werden die dazu passenden Eigenschaften berücksichtigt. Insgesamt ist meist keine gemeinsame Nutzung

Tabelle 3.5: Übersicht der Bewertungen ausgewählter Ansätze

Kriterium	Web-Mashups	Web-Anwendungen	Web-Services	CBSE
Eigenschafts-modellierung	UI, Metriken	Web-Apps	Kein UI	Quellcode
Anforderungs-modellierung	Keine Anforderungen	Web-Apps	Anpassbare Anforderungen	CQML⁺
Integration	EUD-Werkzeuge	Framework	Domänenspezi-fische Lösungen	Framework
Auswertung	Keine Auswertung	Manuell zur Laufzeit	Zur Laufzeit	Framework
Nutzung	Empfehlungen	Feedback	Automatische Auswertung	Auswertung zur Laufzeit

von Modellen für die Belange *Entwicklungszeit* und *Laufzeit* möglich, die eine automatisierte Weiterverwendung festgelegter Qualitätsanforderungen zulässt. Ursache dafür ist vor allem die fehlende Spezifizierung umfassender Entwicklungsprozesse, die auch Nutzeranforderungen der Laufzeit einbeziehen. Diese integrative Herangehensweise ist aus der herkömmlichen Anwendungsentwicklung kaum bekannt und stellt eine in dieser Arbeit behandelte Herausforderung

für das Paradigma kompositer Web-Mashups dar. Viele Eigenschaften erster Eigenschaftsmodelle im Mashup-Bereich decken sich mit denen aus den Web-Services und aus den analysierten Standards. Einige sind allerdings nicht gut für Mashups geeignet, wie etwa die »Skalierbarkeit«. Insbesondere fehlen bei Modellen der Web-Services UI-relevante Qualitätseigenschaften. Diese können allerdings – vor allem im Bereich Usability – aus Modellen für Web-Anwendungen bezogen werden. Die aufgezeigte Stärke der untersuchten Mashup-Ansätze für das Einbeziehen von Qualitätseigenschaften bietet Potenzial zur Ergänzung mit der Expertise im Bereich der Anforderungsmodellierung und -auswertung, die im Bereich Web-Services und der Softwarekomposition auf Code-Ebene stärker etabliert ist. In der Mashup-Plattform CRUISE, die die Basisinfrastruktur für die in dieser Arbeit vorgeschlagenen Konzepte darstellt, werden bisher nur auf Fähigkeiten der Mashup-Komponenten beruhende Vorschläge als rudimentäre Anforderungen in einer Vorstufe des Empfehlungsszenarios ①, vgl. Unterabschnitt 2.2.2, angeboten. Aus dem Fazit der Analyse des Standes der Forschung und Technik ergeben sich somit folgende konzeptionelle Zielstellungen. Aus den einzelnen Qualitätseigenschaften wird ein Referenzmodell erstellt, das eine umfassende und typische Zusammenstellung der relevanten Eigenschaften für den Anwendungstyp Web-Mashup und dessen Bestandteile enthält. Diese sind in eine Struktur einzuordnen, die aus den Anforderungen der Beispielszenarien abgeleitet ist und wichtige Paradigmen zum Aufbau für Qualitätsmodelle aus den untersuchten Standards enthält. Auf diesen Eigenschaften ist ein Anforderungsmodell derart zu konzipieren, dass die modellierten Eigenschaften automatisiert eingebunden, Bedingungen darauf ausgewertet und Adaptionsaktionen ausgelöst werden können. Notwendige Neuerung ist auch ein integrierter Entwicklungs- und Nutzungsprozess, der Empfehlungsverfahren beinhaltet, die das iterative Erstellen – Live-Sophistication – in den Vordergrund stellen.

Kapitel 4

Modellierung von Qualitätseigenschaften für Mashups

Um Qualitätsanforderungen innerhalb eines Mashup-Entwicklungsprozesses oder während des Betriebes einer kompositen Anwendung nutzbar zu machen, bedarf es einer Menge messbarer Qualitätseigenschaften, die für das zu bewertende System oder die Anwendung typisch ist. Formal spezifiziert bilden diese Qualitätseigenschaften ein *Qualitätseigenschaftsmodell*. Während in Softwareentwicklungsprozessen, die Qualitätsmodellierung unterstützen, solche Eigenschaften auf sehr viele unterschiedliche Arten von Softwaresystemen angewendet werden sollen, ist es das Ziel dieser Arbeit, ein für das Anwendungsparadigma *Web-Mashup* maßgeschneidertes Qualitätsmodell zu definieren. Ziel dabei ist insbesondere das Berücksichtigen der Charakteristika von Mashup-Komponenten mit ihrer typischen Granularität in einer Mashup-Plattform. Eine derartige Modellierung ist Basis für die Kontexterfassung, das Monitoring und das Speichern aktueller Ausprägungen von Werten bestimmter Qualitätseigenschaften. Diese Formen der Manipulation bzw. des Auslesens der Wertebelegungen werden teilweise durch die Mashup-Infrastruktur – vor allem die Laufzeitumgebung und das Repository für Komponenten – vorgegeben, bedürfen an vielen Stellen allerdings einer Erweiterung, die in dieser Arbeit vorgenommen wird. Primärer Beitrag des Kapitels ist somit das Aufstellen und das Umsetzen struktureller Anforderungen an die Modellierung von Qualitätseigenschaften in einer Mashup-Infrastruktur. Im resultierenden Metamodell werden sowohl die Erkenntnisse der Analyse existierender Arbeiten als auch die Eignung für die anschließende Definition von Qualitätsanforderungen berücksichtigt. Herausforderung dabei ist das Einordnen dieses Modells in die existierende Modellierungslandschaft der vorgestellten Mashup-Infrastruktur. Eine erste Validierung dieser strukturellen Charakteristika erfolgt durch das Aufstellen typischer Referenzeigenschaften.

Zunächst wird in diesem Kapitel ein Überblick der zentralen semantischen Modelle und ihrer Abhängigkeiten untereinander sowie zu weiteren externen Modellen gegeben. Anschließend erfolgt die detaillierte Vorstellung des Konzeptes zur Modellierung von Qualitätseigenschaften in einer Web-Mashup-Infrastruktur unter Berücksichtigung wichtiger Anforderungen an das Modell. Außerdem werden bestimmte Formen der Nutzung der modellierten Eigenschaften, die neben der Verwendung innerhalb von Qualitätsanforderungen relevant sind, sowie die dafür benötigten Schnittstellen und infrastrukturellen Integrationspunkte präsentiert. Das schließlich vorgestellte Referenzmodell für Qualitätseigenschaften liefert passende Instanzdaten, aus denen sich auch die initial betrachteten Gruppen von Anwendungsszenarien bedienen.

Tabelle 4.1: Überblick der semantischen Qualitätsmodelle

Name	Inhalt
property	Metamodell für Qualitätseigenschaften
property-reference	Referenzmodell für Qualitätseigenschaften
fuzzy	Metamodell für unscharfe Qualitätseigenschaften
fuzzy-reference	Referenzmodell für unscharfe Qualitätseigenschaften
aggregation	Aggregationsmodell für Qualitätseigenschaften
requirement	Qualitätsanforderungen

4.1 Modellüberblick und Abhängigkeiten

Qualitätseigenschaften für Mashup-Komponenten und daraus aufgebaute komposite Anwendungen bilden das Zentrum der Modellierungsgrundlage, die eine vielseitige Verwendung innerhalb von Qualitätsanforderungen in verschiedenartigen Anwendungsszenarien zulässt. Es handelt sich dabei um Modelle, die sich durch die Verwendung von OWL nahtlos in die semantische Beschreibungslandschaft der Referenzinfrastruktur integrieren. Dabei erfolgt eine strukturelle und konzeptionelle Aufteilung in mehrere Verantwortungsbereiche. Grundsätzlich werden in einem Metamodell für Qualitätseigenschaften die strukturellen Charakteristika in allgemeiner Form beschrieben. Das Referenzmodell ergänzt konkrete Eigenschaftstypen für Web-Mashups als Anwendungsdomäne. Für die Konzepte der Beschreibung von Fuzzy-Termen zur Formulierung unscharfer Anforderungen mittels Zugehörigkeitsfunktionen gilt das Separieren in Meta- und Referenzmodell analog. An diesen Kernkomplex knüpfen die Modelle für Qualitätsanforderungen und für die Aggregation von Qualitätseigenschaften auf Anwendungsebene an. Einen Überblick der hier entwickelten semantischen Qualitätsmodelle zeigt Tabelle 4.1. Modelle zur Eigenschaftsmodellierung sind dabei einheitlich folgendem Namensraum zugeordnet:

```
http://mmt.inf.tu-dresden.de/models/quality/property#
```

Den entsprechenden *Basis-URI* erhält man durch Weglassen des Raute-Symbols (#) am Ende des Namensraumes. Er wird zum Auflösen relativer URIs benötigt. Dies gilt analog für das Metamodell für Qualitätsanforderungen, das durch folgenden Namensraum repräsentiert ist:

```
http://mmt.inf.tu-dresden.de/models/quality/requirement#
```

Abbildung 4.1 zeigt die Abhängigkeiten der semantischen Modelle untereinander. Schwarz dargestellt sind die in dieser Arbeit neu entwickelten semantischen Modelle. Die durch die Pfeile visualisierte Abhängigkeit zwischen zwei Modellen bedeutet, dass die semantischen Begriffe und Beziehungen eines Modells in einem anderen, abhängigen Modell sichtbar sind und verwendet werden können, um beispielsweise mit den zusätzlich verfügbaren semantischen Ressourcen weiteres Wissen zu modellieren. Zu diesen Abhängigkeiten zählt auch die Verwendung eines Metamodells in einem Referenzmodell, wie es bei *fuzzy* und *property* jeweils der Fall ist. Die transitive Verwendung der Abhängigkeiten ist möglich, wie beispielsweise die Nutzung von *property* innerhalb von *fuzzy-reference* über *fuzzy*.

Tabelle 4.2: Modifizierte und importierte semantische Modelle

Namensraum	Inhalt
http://qudt.org/1.1/vocab/unit	QUDT-Einheiten
http://mmt.inf.tu-dresden.de/models/smcdl#	Mashup-Komponentenmodell
http://mmt.inf.tu-dresden.de/models/mcm#	Mashup-Kompositionsmodell

Grau dargestellt und mit gestrichelten Pfeilen verbunden sind zum einen importierte, existierende Domänenmodelle wie *unit*, zum anderen *component* und *application* als semantische Repräsentation der Mashup-Komponenten (SMCDL) und Mashup-Anwendungen (MCM), die die hier beschriebene Qualitätsinfrastruktur nutzen und dafür angepasst wurden, vgl. Tabelle 4.2. Es wird ein Modell zur Beschreibung von Maßeinheiten des Projektes Quantities, Units, Dimensions and Data Types Ontologies (QUDT) [Hod+14] verwendet, um den Qualitätseigenschaften sinnvolle und vor allem durch Maschinen verarbeitbare Einheiten zuzuweisen. Insbesondere bei der Umrechnung verschiedener Einheiten derselben Größe bieten die QUDT-Modelle Unterstützung. Durch das Abstützen auf das Internationale Einheitensystem (SI) wird außerdem der standardisierte Austausch mit Modellen anderer Autoren gewährleistet. Die strukturellen Abhängigkeiten jenseits von *unit* wurden in der Abbildung zur Vereinfachung weggelassen.

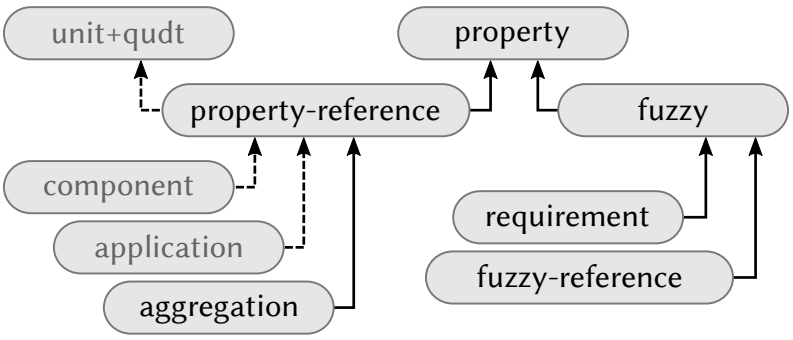

Abbildung 4.1: Abhängigkeiten der semantischen Qualitätsmodelle und Referenzmodelle untereinander sowie zu externen Domänenmodellen

Die vorgestellten Modelle bilden ein *Kontinuum*, das durch den Zugriff über die Schnittstellen – vgl. Abschnitt 4.5 –, das Vorhalten sowie die Ausführung in einem Laufzeitcontainer, eine einheitliche Anfrage über alle Modelle gestattet. Gewährleistet wird dies durch die Modellverwaltung mit einem semantischen Framework wie *Jena*[1] im CoRe. Die Zuordnung der Qualitätseigenschaftswerte, beispielsweise zu konkreten Mashup-Komponenten über die SMCDL, wird in Unterabschnitt 4.5.1 beschrieben. Unter den vorgestellten Modellen betrachtet dieses Kapitel im Detail vor allem die für die Qualitätseigenschaften, die Fuzzy-Modellierung und die Aggregation. Die Anforderungsmodellierung wird speziell im nächsten Kapitel aufgegriffen und bezieht die Ergebnisse dieses Kapitels ein.

[1] Jena, Framework für Anwendungen des semantischen Webs: `jena.apache.org`

4.2 Anforderungen an das Eigenschaftsmodell

Sowohl die vorgestellten Anwendungsszenarien als auch die definierten Forschungsziele und die zugrundeliegende Infrastruktur definieren Anforderungen an die Art und Weise der Modellierung von Qualitätseigenschaften. Dabei werden nicht nur *inhaltliche* Aspekte des Referenzmodells berücksichtigt, sondern auch der *strukturelle* Aufbau des Metamodells sowie die Schnittstellen zur geplanten Modellnutzung und -interaktion. Die hier beschriebenen Anforderungen dienen somit der Bewertung der Eignung der Qualitätsmetamodelle sowie der Einordnung der Schwerpunkte des in Abschnitt 4.6 präsentierten Referenzmodells.

Anreicherung mit unscharfen Termen: Das Referenzmodell soll *Fuzzy-Sets* auf den festgelegten Qualitätseigenschaften unterstützen. Dafür ist es erforderlich, die Zuordnung entsprechender Fuzzy-Terme zu den Eigenschaften vorzunehmen. Im Metamodell sind dafür die geeigneten Modellierungsmöglichkeiten vorzusehen. Die zusätzliche Möglichkeit der Nutzer, unscharfe Begriffe mit den gewünschten Eigenschaften innerhalb von Qualitätsanforderungen verwenden zu können, trägt zur angestrebten Vergrößerung der Zielgruppe während der Nutzung von Mashup-Plattformen bei, vgl. Unterabschnitt 1.2.2. Als Grundlagen zur Verwendung umgangssprachlicher Formulierungen innerhalb der automatisierten Verarbeitung von Qualitätsanforderungen dienen die in Abschnitt 4.4 detailliert betrachteten Konzepte.

Belangorientiertheit: Strukturell soll das Modell verschiedene Belange der Eigenschaftsträger – beispielsweise das UI oder angebundene Dienste – unterstützen, ohne eine Variantenexplosion einzelner Eigenschaften zu verursachen. Die Facettierung in solche *Concerns*, die bei vielen Eigenschaften auftauchen, dient der Verbesserung der Übersichtlichkeit bei der Instanzmodellierung, bei der Spezifizierung und bei der Nutzung innerhalb von Qualitätsanforderungen. Diese Anforderung betrifft sowohl das Metamodell, vgl. Unterabschnitt 4.3.1, als auch das Referenzmodell für Qualitätseigenschaften, in dem die Eigenschaften mit Instanzdaten angereichert werden.

Aggregierbarkeit (Eigenschaftswerte): Insbesondere für die Wertebelegung der Qualitätseigenschaften von Mashup-Komponenten sind zwei Aggregationsszenarien relevant. Zunächst können Qualitätseigenschaften *inhaltlich in einer Hierarchie* aggregiert werden, um beispielsweise dem Nutzer eine verallgemeinerte Sicht zu bieten. Hierbei werden etwa im Bereich der Sicherheitsbewertungen verschiedene Teileigenschaften zu einer singulären Bewertung zusammengefasst. Außerdem soll durch Aggregation der Komponentenbewertungen *auf Anwendungsebene* die Vergleichbarkeit kompositer Mashups verbessert werden. Hierbei ist auf automatisierbare Aggregation zu achten, die durch semantisch beschriebene Datentypen sowie durch die Verwendung einheitlicher, verbreiteter Modelle – wie QUDT – erreicht werden kann. In beiden Fällen sind geeignete Aggregationsvorschriften vorzusehen, um aussagekräftige Gesamtbewertungen zu erhalten.

Erweiterbarkeit: Das Modell soll gewährleisten, dass später weitere Eigenschaften hinzugefügt werden können. Die Technologieauswahl und eine einheitliche strukturelle Modellierung der Eigenschaften unterstützen dies. Um das Vorgehen bei der Ergänzung, Einordnung und Berücksichtigung neuer Eigenschaften möglichst einfach zu gestalten, ist

ein vereinheitlichter Prozess zu spezifizieren. Als Ergänzung im Sinne der *Anpassbarkeit* des Referenzmodells soll es zudem ermöglicht werden, Eigenschaften beispielsweise für domänenspezifische Einsatzszenarien wegzulassen, d. h. das Modell im Sinne von Profilen einzuschränken. Das Modell kann somit für andere Szenarien oder Anwendungssysteme nutzbar gemacht werden.

Maschinelle Verarbeitbarkeit: Hier spielt sowohl die nahtlose Integrierbarkeit in bestehende Modelle wie SMCDL und MCM, vgl. Abschnitt 4.1, als auch die automatisiert auswertbare Modellierung der Inhalte mittels semantischer Beschreibungen im Auswertungsprozess von Qualitätsanforderungen in einer Infrastruktur für komposite Web-Mashups eine Rolle. Um die Automatisierung weiter zu verbessern, soll jede Eigenschaft Vorgabewerte erhalten, damit zumindest ein Basisniveau an Vergleichbarkeit – etwa in einem Empfehlungsprozess mit vielen Kandidaten – gewährleistet ist.

Eignung zur Anforderungsdefinition: Primärer Nutzungszweck des Modells für Qualitätseigenschaften ist die Verwendung ihrer Wertebelegung in Qualitätsanforderungen. Neben der grundlegenden maschinellen Verarbeitbarkeit muss sichergestellt werden, dass die Versorgung der Eigenschaften mit Werten – durch Messung oder Bereitstellung – gewährleistet ist. Dies macht die Unterscheidung in verschiedene Typen notwendig. Außerdem muss die Verwaltung sichergestellt werden, die etwa bei der Aggregation von vielen Einzelmessungen zu einer Gesamtbewertung für eine Eigenschaft bei statistisch ermittelten Größen entscheidend ist. Die Einteilung der Eigenschaften in Typen der Wertermittlung ist unter anderem die Voraussetzung für die Zuordnung von Kontextsensoren.

Angemessenheit (Anwendungstyp): Das Referenzmodell soll charakteristische Qualitätseigenschaften für Web-Mashups enthalten. Hierbei ist zu berücksichtigen, welche typischen Bestandteile bzw. Rollen in einer Mashup-Infrastruktur Träger solcher Eigenschaften sein können. Die Abgrenzung zu anderen Typen von Anwendungs- und Softwaresystemen wird durch die Gegenüberstellung der vorgeschlagenen Eigenschaften mit denen aus existierenden Arbeiten vorgenommen.

Eigenschaftsträger: Qualitätseigenschaften des Referenzmodells sollen bestimmten Trägern zugeordnet bzw. danach abgegrenzt werden können. Das betrifft vor allem Anwendungen und Komponenten der Mashups als Domänenobjekte. Grundsätzlich soll die Infrastruktur zur Modellierung und Auswertung ebenfalls für Laufzeitumgebungen, Betriebssysteme der ausführenden Geräte mit Sensorik sowie Benutzer und Kontextmodelle anwendbar sein. Entitäten, die derartige Eigenschaftsträger sein können, wurden bereits in Abschnitt 2.3 näher beschrieben. Ziel ist es dabei außerdem, die Redundanz gemeinsam genutzter Eigenschaften durch intelligente Importe auf Metamodellebene zu reduzieren.

Die ersten sechs Anforderungen sind der Metamodellierung zuzuordnen, die die Struktur und die Charakteristika für Qualitätseigenschaften und deren beabsichtigte Nutzungszwecke festlegt. Die verbleibenden zwei Anforderungen betreffen das Referenzmodell und seine Passgenauigkeit innerhalb des Typs von Anwendungssystemen, die durch das Paradigma kompositer Web-Mashups bestimmt wird. Aus den Anforderungen an die Metamodellebene ergibt sich eine Facettierung der Modellstruktur, die für die zu spezifizierenden Qualitätseigenschaften gelten soll. In den folgenden Abschnitten werden diese Strukturierungsmöglichkeiten im Detail

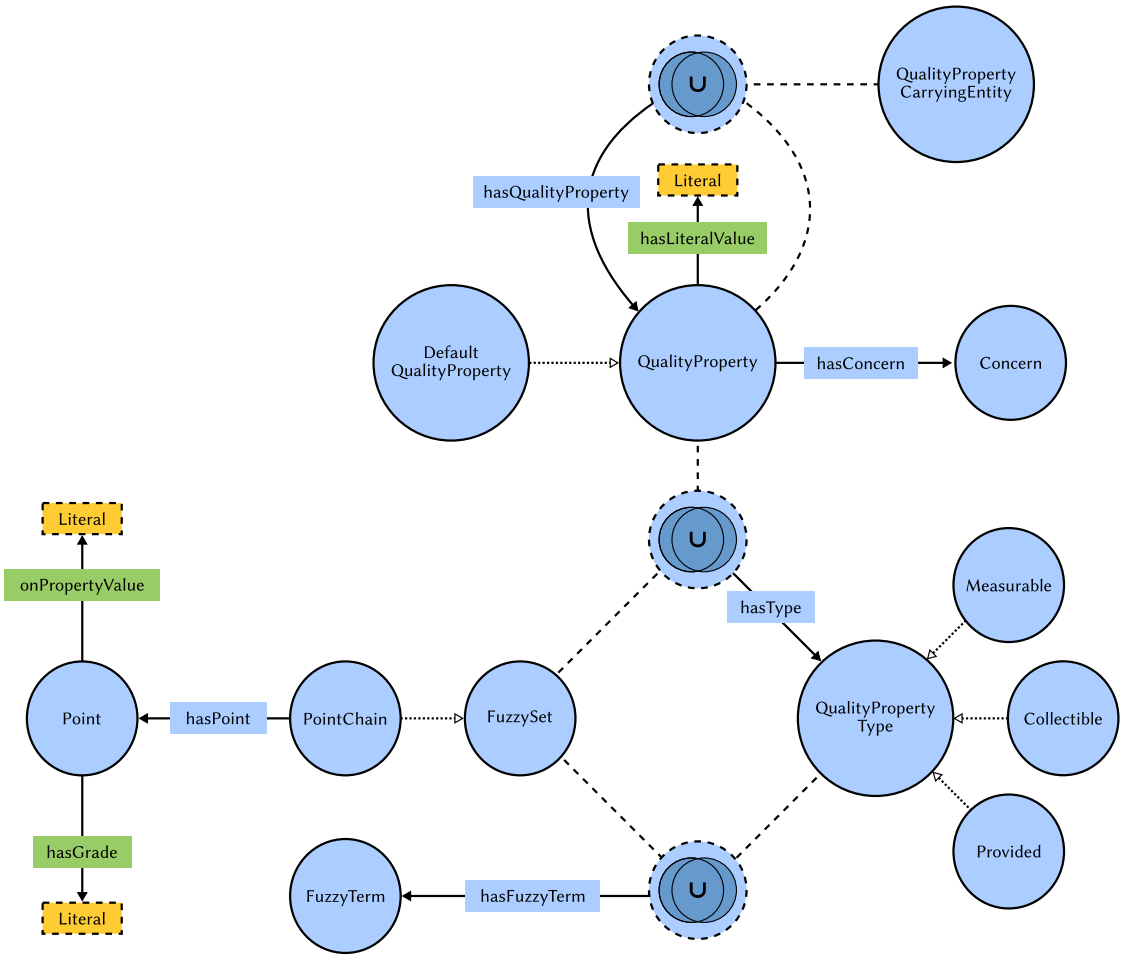

Abbildung 4.2: Metamodell der Qualitätseigenschaften für komposite Web-Mashups

vorgestellt. Es handelt sich dabei vor allem um *Concerns*, die bestimmte Aspekte der Eigenschaftsträger bezüglich einer Qualitätseigenschaft einfach adressierbar machen, *Fuzzy-Sets*, die mit umgangssprachlichen Termen den Zugang zu den Eigenschaften erleichtern, und *Typen der Wertermittlung* zur angemessenen Integration in den Entwicklungsprozess für komposite Web-Mashups. Anschließend daran erfolgt die Behandlung der Anforderungen zur Nutzung des Eigenschaftsmodells, unter anderem während der *Aggregation* auf Anwendungsebene, sowie das Vorstellen des *Referenzmodells* für komposite Web-Mashups.

4.3 Metamodell zur Strukturierung von Qualitätseigenschaften

Das Metamodell für Qualitätseigenschaften wird in Abbildung 4.2 in der Visual Notation for OWL Ontologies (VOWL) [Loh+16] gezeigt. *Quality Propertys* sind dabei entsprechenden *Entitys* (oben rechts) zugeordnet. Neben Mashup-Komponenten und -anwendungen können das

auch Nutzer und Geräte mit Laufzeitumgebungen und Sensorik sein, vgl. auch Anforderungsträger und zugehörige Anforderung an das Modell aus dem vorhergehenden Abschnitt. Über *hasQualityProperty* werden Qualitätseigenschaften zudem rekursiv zugeordnet. Qualitätseigenschaften besitzen – vor allem auf Blattebene des Rekursionsbaumes – literale Werte über *hasLiteralValue*. In der Abbildung fällt auf, dass *hasLiteralValue* über ein Symbol mit der Vereinigungsmenge ∪ mit mehreren *Domains* verbunden ist. Es taucht noch weitere Male in diesem und anderen Modellen dieser Arbeit auf und zeigt die Zulässigkeit verschiedener Klassen als Domain in der OWL-Zuordnung des Definitionsbereichs (Domain) zum Wertebereich (Range) an. *Default Quality Property Values* werden benötigt, um beispielsweise bei fehlenden Angaben über Vorgabewerte Mashup-Komponenten vergleichbar zu machen. Dies ist besonders bei der Empfehlung und bei der Berechnung von aggregierten Werten für Qualitätseigenschaften auf Anwendungsebene erforderlich. Optional auf Qualitätseigenschaften anwendbare *Concerns* bieten eine zusätzliche Facettierung innerhalb der Wertebelegung und Anforderungen. Sie werden in Unterabschnitt 4.3.1 näher erklärt. Die Semantik einer Qualitätseigenschaft wird vorwiegend durch ihren Typ bestimmt. Das Referenzmodell in Abschnitt 4.6 bietet eine umfangreiche Auswahl von Beispielen für Typen von Qualitätseigenschaften aus dem Anwendungsparadigma Web-Mashups. Unten rechts in Abbildung 4.2 werden mögliche Typen nach der Art der Ermittlung der Wertebelegung unterteilt. Diese Unterteilung ist Gegenstand von Unterabschnitt 4.3.2. Der untere linke Teil des Metamodells beschäftigt sich mit der Möglichkeit, umgangssprachliche *Terme* innerhalb von Anforderungen auf Qualitätseigenschaften verwenden zu können. Eine ausführliche Beschreibung der Modellelemente gibt Abschnitt 4.4. Im Konzept zur Adressierung von Qualitätseigenschaften innerhalb von Qualitätsanforderungen in Unterabschnitt 5.3.1 wird eine Abstraktion von *Quality Property* auf *Feature* vorgenommen, um Anforderungen universell einzusetzen. In diesem Kapitel wird jedoch zunächst auf den spezifischen Aufbau und die Beziehungen der Qualitätseigenschaften eingegangen.

4.3.1 Concerns

Qualitätseigenschaften können sowohl im Kontext des UI als auch im Kontext ihrer zu verarbeitenden Daten oder bezüglich ihrer angebundenen Web-Services betrachtet werden. Um eine Variantenexplosion für einige wichtige Ausprägungen dieser *Belange* (englisch: *concerns*) bei der Modellierung in Kombination mit den tatsächlichen Eigenschaften zu vermeiden, finden sie als strukturelles Merkmal Einzug in das Metamodell für Qualitätseigenschaften. Im Zusammenhang mit den typischen Eigenschaften der Kompositionsfragmente bei Web-Mashups wird folgendes Referenzmodell für Concerns vorgeschlagen, das um weitere Belange erweiterbar ist. Die Verortung dieser Referenz-Concerns am Beispiel kommunizierender Mashup-Komponenten als Black-Boxes mit UI zeigt Abbildung 4.3.

API: Dem Anwendungsentwickler steht innerhalb des Mashup-Paradigmas nur das API der Komponente bei der Entwicklung mit Black-Box-Bestandteilen zur Verfügung, um durch Kommunikation den Daten- und Kontrollfluss zu realisieren. Diese insbesondere auf Komponentenebene relevanten Schnittstellen können als Concern innerhalb von Qualitätseigenschaften genutzt werden.

UI: Eine weitere von außen sichtbare Schnittstelle von Mashup-Komponenten ist neben dem Komponenten-API das UI, welches ebenfalls als Concern angeboten wird. Im Gegensatz

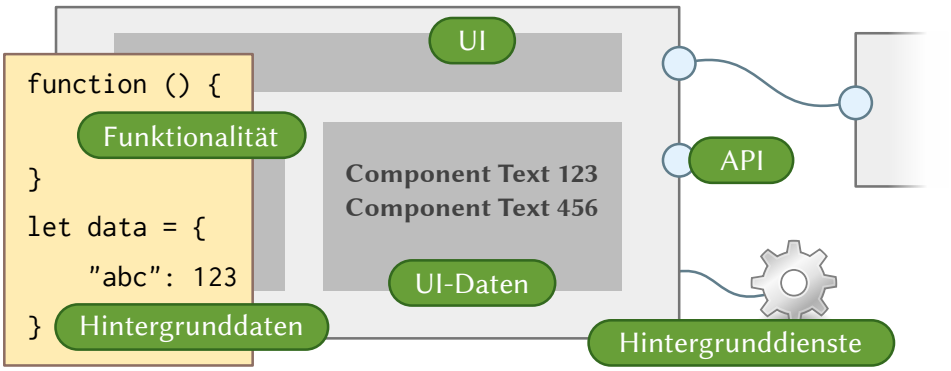

Abbildung 4.3: Concerns der Qualitätseigenschaften bei Web-Mashups

zum API, das zur Kommunikation von Softwarekomponenten untereinander dient, ist der Nutzer der Kommunikationspartner des Komponenten-UI. Da Mashup-Anwendungen grundsätzlich auch mit einem UI ausgestattet sind, ist der Concern auch für sie geeignet. Bei reinen Service-Komponenten hat der UI-Concern keine Anwendungsmöglichkeit für Qualitätseigenschaften, da keine Nutzerschnittstelle angeboten wird.

Funktionalität: Funktionale Fähigkeiten (englisch: *capabilities*), wie etwa die Berechnung einer Route innerhalb einer Kartenkomponente, stehen im Kontext dieses Concerns. Bei Anwendungen zählt dazu auch die Mashup-interne Kommunikation zwischen den Komponenten. Im Gegensatz zum Concern API werden hier auch Funktionalitäten betrachtet, die nicht über eine Schnittstelle angeboten, sondern ggf. implizit erbracht werden. Darüber kann auch eine funktionale Dekomposition berücksichtigt werden.

Hintergrunddienste: Viele Mashup-Komponenten setzen zur Erbringung ihrer Dienstleistungen weitere Web-Services ein. Dieser Concern bezieht Qualitätseigenschaften – etwa das *Antwortverhalten* der angebundenen Dienste betreffend – ein.

UI-Daten: Im UI visualisierte Daten sind der Gegenstand dieses Concerns. Insbesondere die Präsentation der in der Komponente bzw. Anwendung zur Verfügung stehenden Daten werden hierbei – wie etwa bei der *Aktualität* der präsentierten Daten – betrachtet.

Hintergrunddaten: Dieser Concern betrifft Daten, die innerhalb einer Mashup-Komponente bzw. -anwendung genutzt werden, deren Anzeige im UI jedoch nicht im Vordergrund steht. Der Fokus liegt hierbei auf der Verarbeitung dieser Daten, wie beispielsweise in der Qualitätseigenschaft *Sicherheit*.

Im Metamodell für Qualitätseigenschaften zeigt die Zuordnung über *hasConcern* die für eine Eigenschaft mit bestimmtem Typ anwendbaren Concerns an. Nicht für alle Qualitätseigenschaften ist eine Zuordnung zu Concerns sinnvoll. Ohne die optionale Angabe des Concerns bezieht sich eine Anforderung mit einem bestimmten Eigenschaftstyp auf den Träger – beispielsweise die Mashup-Komponente – im Allgemeinen.

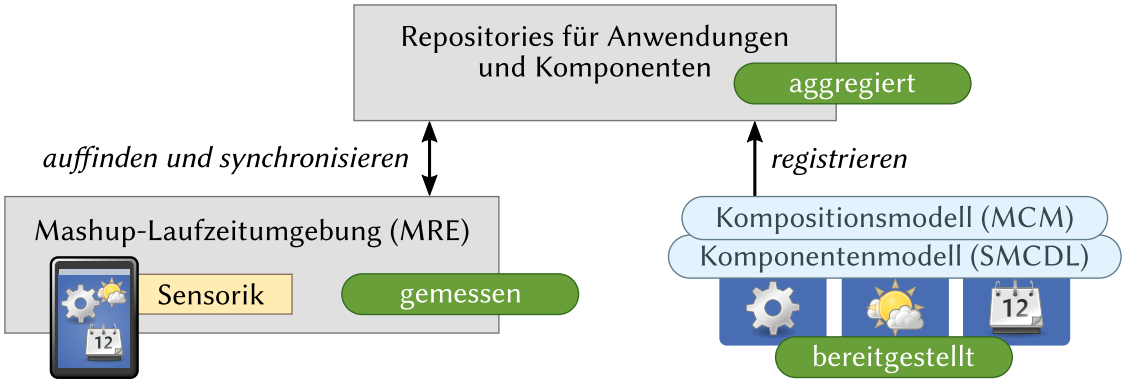

Abbildung 4.4: Wertermittlung von Qualitätseigenschaften in der Mashup-Infrastruktur

4.3.2 Typen der Wertermittlung für Qualitätseigenschaften

Zur Auswertung von Anforderungen, die Bedingungen über Qualitätseigenschaften enthalten, ist es zwingend erforderlich, dass die genutzten Eigenschaften mit Werten belegt sind. Die für das Durchführen von Vergleichen benötigte Wertebelegung ist dabei durch die Häufigkeit der Änderung, die Rolle der setzenden Entität innerhalb der Infrastruktur und die Methodik des Festlegens durch einen Entwickler, Nutzer oder die Sensorik charakterisiert. Hauptsächlich kann zwischen zur Laufzeit ermittelten und im Komponentenmodell vorhandenen Werten unterschieden werden. Der zweite Fall unterscheidet sich weiter in Eigenschaften, die *vom Autor eines Kompositionsfragmentes vorgegeben* werden, und Eigenschaften mit *gesammelten* Werten, die über eine längere Zeit aus Einzelwerten aggregiert wurden, beispielsweise der Durchschnitt vieler Nutzerbewertungen. Abbildung 4.4 zeigt die Verortung der drei Typen der Wertermittlung in der abstrakten Mashup-Infrastruktur. Die folgende Übersicht gibt Aufschluss darüber, wer den Eigenschaftswert zur Verfügung stellt, zu welchen Gelegenheiten bzw. wie oft der Wert ermittelt wird und wo bzw. wie die Wertebelegungen verwaltet oder persistiert werden. Die Beispieltypen der Qualitätseigenschaften stammen aus dem in Abschnitt 4.6 vorgestellten Referenzmodell.

Bereitgestellt: Statische Werte, zu welchen auch klassische Metadaten – wie etwa der Preis oder Kontaktdaten – gehören, gibt der Autor eines Kompositionsfragmentes zur Entwicklungszeit an. Sie werden oft einmalig bereitgestellt (englisch: *provided*) und ändern sich nie oder nur selten. Bei Mashup-Komponenten werden statisch bereitgestellte Werte für Qualitätseigenschaften im Komponentenmodell gespeichert.

Aggregiert: Einige Eigenschaften, wie etwa die Bewertung der Zufriedenheit durch Nutzer oder die Verfügbarkeit, sind über ein singuläres Bereitstellen oder eine Einzelmessung wenig aussagekräftig. Sie müssen daher über mehrere statistische Messwerte oder Einzelangaben gesammelt (englisch: *collected*) und aggregiert werden. Mit einer großen Nutzeranzahl werden darüber auch die Effekte von Ausreißern minimiert. Die Häufigkeit des Bereitstellens variiert hier von einer Aggregation regelmäßig durchgeführter Messungen bis hin zur gelegentlichen, nutzergetriebenen Bereitstellung der Einzelwerte, wie es bei Eigenschaften der Fall ist, die durch Feedback zustande kommen.

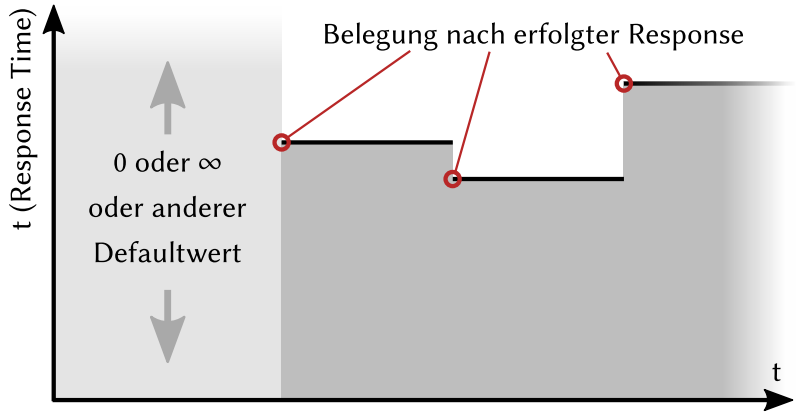

Abbildung 4.5: Aktualisierung der Wertebelegung bei gemessenen Eigenschaften am Beispiel des Typs *ResponseTime*

Messbar zu Laufzeit: Für Qualitätseigenschaften, die nicht im Vorfeld bereitgestellt, sondern durch das Laufzeitsystem gemessen (englisch: *measured*) werden, ändert sich die Wertebelegung in der Regel rascher. Sie sind dadurch für Überwachungsaufgaben und die Adaptivität zur Laufzeit von Anwendungen relevant. So stehen beispielsweise die aktuelle Antwortzeit von Hintergrunddiensten oder der Speicherverbrauch von Komponenten im Betrieb für die Anforderungsauswertung zur Verfügung.

Besonders bei messbaren Eigenschaften ist die Häufigkeit der Aktualisierung der tatsächlich auslesbaren Wertebelegung stark abhängig vom Eigenschaftstyp. Bei sehr hoher Änderungsfrequenz – wie etwa beim Auslesen von Geopositionen aus einem Sensor – bietet sich eine *Taktung* an. Andere Typen begnügen sich mit einer *ereignisgetriebenen* Aktualisierung. In Abbildung 4.5 wird die Änderung der Wertebelegung bei in der Laufzeitumgebung gemessenen Typen von Qualitätseigenschaften gezeigt. Sobald der Eigenschaftsträger – beispielsweise eine Komponente – in der Laufzeitumgebung ausgeführt wird, gilt zunächst ein Vorgabewert (englisch: *default*). Diese Belegung ändert sich, sobald eine Messung stattgefunden hat. Ab diesem Zeitpunkt wird der Messwert als Belegung der Qualitätseigenschaft gesetzt. Er behält seine Gültigkeit, bis ein neuer Messwert ermittelt wird. Das führt dazu, dass alle Anfragen an den Messwert – etwa durch Auswertung von Qualitätsanforderungen ausgelöst – zwischen zwei Messungen denselben Eigenschaftswert ergeben. Die zu den abstrakten Typen der Wertermittlung gehörenden Schnittstellen zum Lesen und Schreiben von Eigenschaftswerten in der Mashup-Plattform werden in Abschnitt 4.5 bzw. in Unterabschnitt 7.4.3 näher beschrieben.

4.4 Unscharfe Eigenschaftswerte mit Fuzzy-Mengen

Unter Nutzung des geeigneten Modells für Qualitätseigenschaften kann das Spezifizieren von Qualitätsanforderungen in formalisierter Art und Weise über *Bedingungen* erfolgen. Klassische Bedingungen lassen jedoch nur *Identitätsabfragen* oder einfache *Bereichsabfragen*, beispielsweise über den Operator $<$ (kleiner als), zu. Insbesondere bei der Kombination mehrerer Bedingungen entstehen so oft ungewollte Ausschlusskriterien, die ein iteratives Filtern erschweren. Au-

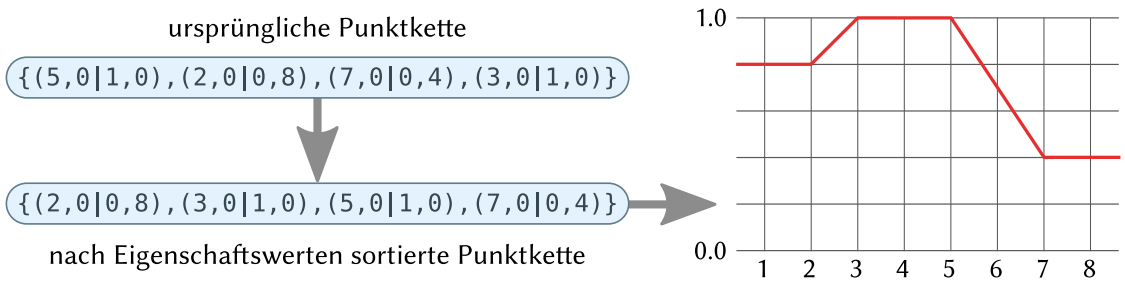

Abbildung 4.6: Schrittweise Konstruktion einer Zugehörigkeitsfunktion für Fuzzy-Mengen auf Basis einer Punktkette

ßerdem zerfällt die Menge der zu bewertenden Kandidaten in nur jeweils zwei Klassen mit dem binären Erfüllungsgrad der Bedingung als einzigem Unterscheidungskriterium. Sollen bewertete Mashup-Komponenten in eine differenzierte Reihenfolge gebracht werden, wie es etwa in einem Empfehlungsprozess sinnvoll ist, fehlt hier die rechnerische Grundlage. Das Konzept der Qualitätseigenschaften wird daher um *Unschärfe* (englisch: *fuzziness*) erweitert, um sowohl eine unscharfe Anforderungsspezifikation als auch eine feingranular abgestufte Bewertung zu ermöglichen. Dieser Abschnitt stellt die notwendigen Erweiterungen im Modell sowie die Anwendungsmöglichkeiten und Rahmenbedingungen vor. Das Metamodell für Qualitätseigenschaften erfährt somit eine Erweiterung um Fuzzy-Konzepte, vgl. unten links in Abbildung 4.2. Mit der hinzukommenden Fuzzy-Modellierung wird vor allem das Ausdrücken von *Kompromissen* durch das gemeinsame Verwenden von Bedingungen auf verschiedenen Qualitätseigenschaften möglich. Ziel hierbei ist die maschinelle Unterstützung unscharfer Terme wie *möglichst niedrig*, *gut* oder *kostenlos* aus dem Empfehlungsszenario in Unterabschnitt 2.2.2. Neben der unmittelbaren Unterstützung eines menschlichen Anforderungsautors bietet dieses Konzept auch Vorteile in der automatisierten Aufbereitung vorproduzierter, historischer oder aufgezeichneter Texte, die umgangssprachliche Anforderungen enthalten.

4.4.1 Zugehörigkeitsfunktionen mit Punktketten

Das unscharfe Spezifizieren von Anforderungen ist ein komfortabler Weg zur Erhöhung des Abstraktionsgrades bei gleichzeitigem Verbergen detaillierten Domänenwissens in einem nutzergetriebenen Entwicklungsprozess. *Unscharfe Mengen* (Fuzzy-Mengen, englisch: *fuzzy sets*) [Zad65] liefern dafür eine geeignete formale Grundlage. Demnach können *linguistischen Variablen*, wie etwa dem *Preis*, verschiedene *Terme* in natürlicher Sprache zur Beschreibung der Ausprägung – beispielsweise *günstig* oder *sehr teuer* – zugeordnet werden, ohne einen konkreten skalaren Vergleichswert zu nennen. Die linguistischen Variablen entsprechen in dieser Arbeit konzeptionell den Typen der Qualitätseigenschaften. Jedem der *Terme* ist eine *unscharfe Menge* zugeordnet, um eine automatisierte Auswertung zu gewährleisten. Solche *Fuzzy-Sets* definieren den *Zugehörigkeitsgrad* (englisch: *grade of membership*) nicht binär wie klassische Mengen. Stattdessen beschreiben sie *Zugehörigkeitsfunktionen* (englisch: *membership functions*) mit Werten zwischen 0 und 1. Im Allgemeinen können beliebige Funktionen mit einem Wertebereich von $[0, 1]$ als solche Zugehörigkeitsfunktionen dienen. Klassische Mengen mit binärer Zugehörigkeit sind somit ein Spezialfall unscharfer Mengen.

Im Sinne der effizienten Auswertung und der Speicheroptimierung werden typischerweise möglichst einfache Funktionen mit geradlinigen Abschnitten wie etwa Dreiecks- und Trapezformen bevorzugt. Daher werden vor allem Fuzzy-Sets mit Zugehörigkeitsfunktionen verwendet, die als linear verbundene *Punktketten* beschrieben werden können, vgl. auch verwandte Arbeiten in Abschnitt 3.3. Jeder verkettete Punkt weist einem bestimmten Wert einer Qualitätseigenschaft einen Zugehörigkeitsgrad zu. Die Funktion für den gesamten Definitionsbereich ergibt sich durch Sortieren der enthaltenen Punkte nach ihrem Eigenschaftswert. Mehrere unterschiedliche Punkte mit demselben Eigenschaftswert sind aufgrund der dadurch entstehenden Ambivalenz nicht erlaubt. Durch lineares Verbinden der Punkte können somit die Funktionswerte für zwischen den Eckpunkten liegende Eigenschaftswerte eindeutig ermittelt werden. Die Punkte mit den jeweils äußersten Eigenschaftswerten verlängern den Zugehörigkeitsgrad auf ihrer Seite ins Unendliche. Abbildung 4.6 zeigt den Konstruktionsvorgang einer solchen Funktion anhand einer zunächst unsortierten Beispielmenge von Punkten. Resultierende Punktketten, die durch verbundene Eckpunkte gebildet werden, bieten eine für viele Zwecke ausreichende Komplexität der Funktionen. Sie zeichnen sich durch einen geringen Aufwand beim Spezifizieren und Auswerten aus.

Die vorgestellte Konstruktionsweise ist vor allem für Qualitätseigenschaften mit kontinuierlichen Datentypen – beispielsweise *Float* (gebrochene Zahlen) – geeignet. Grundsätzlich sind diskontinuierliche Werte für die Funktionen unproblematisch, da bei der Auswertung nur gültige Eigenschaftswerte abgefragt werden. Ungenutzte Zwischenwerte, die durch die Funktion bereitgestellt werden, sind dabei unschädlich für den Auswertungsvorgang. Beispiele für diskontinuierliche Datentypen sind Zeichenketten (*Strings*) oder feste Werte in einer Aufzählung (*Enumerations*). Unterschiede ergeben sich später in der Werkzeugunterstützung beim Erstellen und Bearbeiten von Funktionen für Fuzzy-Sets, siehe Unterabschnitt 7.5.2. Um bei Termen wie *möglichst gering* eine optimale Sortierung über einen nicht näher begrenzten Definitionsbereich einer Qualitätseigenschaft zu erhalten, bieten sich auch Potenzfunktionen oder Funktionen auf Basis des Arkuskotangens an. Mit $x \geq 0$ als Definitionsbereich und $0 \leq f(x) \leq 1$ als Wertebereich ergeben $f(x) = (x+1)^{-1}$ oder $f(x) = \frac{2}{\pi} \operatorname{arccot} x$ geeignete Zugehörigkeitsfunktionen. Sie stellen durch ihr monotones Gefälle die korrekte Sortierung nach fallenden Eigenschaftswerten x sicher. Im Metamodell genügt in dieser Arbeit die *Point Chain* als Ausprägung von *Fuzzy-Sets*. Da nicht jeder *Fuzzy-Term* auf alle *Quality Property Types* sinnvoll anzuwenden ist, wird über *hasFuzzyTerm* die Zuordnung sinnvoller Terme zu den Typen vorgenommen.

4.4.2 Nutzerdefinierte Konfiguration von Termen

Beispielfunktionen, die auf Fuzzy-Termen von Qualitätseigenschaften gebildet wurden, zeigt Abbildung 4.7. Beachtenswert ist, dass sich verschiedene Funktionen im Definitionsbereich der Qualitätseigenschaft überlappen können, siehe *gering* und *hoch*. Praktisch bedeutet das, dass für einen konkreten Eigenschaftswert mehrere Terme mit einer Zugehörigkeit verschieden von 0 zutreffend sind. Funktionen zu entsprechenden Termen werden initial als *Vorgabefunktionen* definiert. Sie stellen anschließend die Basis der nutzerdefinierten Konfiguration dar. Eine solche Konfiguration nach Nutzerprofilen ist auf der rechten Seite der Abbildung zu sehen. Neben einer *Verschiebung* entlang der Abszisse (blau) kommt eine *Streckung* bzw. *Stauchung* (grün) zum Verändern des Flankenanstieges als einfache Transformation der Zugehörigkeitsfunktion in Betracht. Solche bedarfsgerechten Anpassungen werden als alternative Funktionen neben

der Vorgabefunktion im Instanzmodell repräsentiert. Sie stehen dem beteiligten Nutzer somit als konfigurierbare Vorgabeeinstellungen, die subjektiv interpretiert werden können, zur Verfügung. Funktionsanpassungen und ihre Neudefinition können mittels eines durch die Mashup-Plattform angebotenen Editors vorgenommen werden, der in Unterabschnitt 7.5.2 vorgestellt wird. Neben der nutzergetriebenen Anpassung der Zugehörigkeitsfunktionen von Termen sind in bestimmten Fällen auch noch andere – ggf. automatisierte – Formen der Adaption notwendig. Ein Beispiel dafür sind einseitig oder beidseitig offene Skalen, wie etwa beliebig hohe oder unendlich große Zahlenwerte beim Eigenschaftstyp *Response Time*.

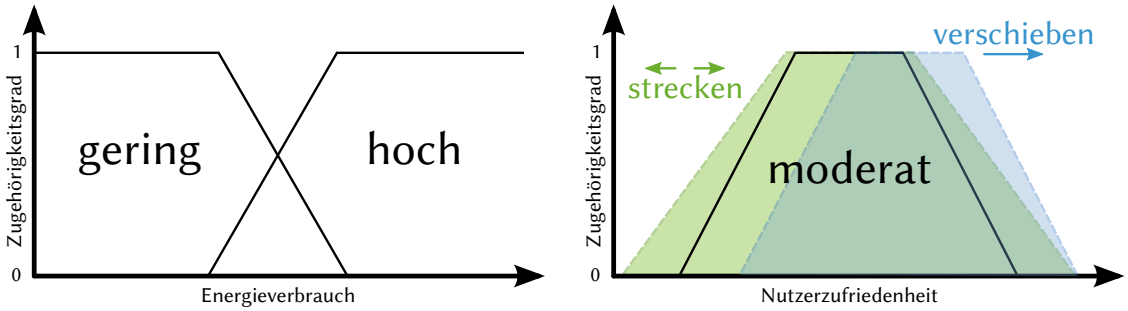

Abbildung 4.7: Konfiguration von Zugehörigkeitsfunktionen

4.5 Nutzung von Qualitätseigenschaften in der Mashup-Plattform

Instanzen des vorgestellten Metamodells für Qualitätseigenschaften dienen zum Speichern von Wertebelegungen dieser Eigenschaften im Kontext der in der Mashup-Plattform auftretenden Eigenschaftsträger, vor allem der Kompositionsfragmente. Die Art der Nutzung wird durch die Form der Wertermittlung des Eigenschaftstyps bestimmt, vgl. Unterabschnitt 4.3.2. Dieser Abschnitt beschreibt die Integration der Eigenschaftswerte in die Mashup-Plattform sowie dafür benötigte Schnittstellen der Artefakte und beteiligte Rollen.

4.5.1 Integration bereitgestellter Eigenschaften

Durch Autoren von Kompositionsfragmenten bereitgestellte Werte für Qualitätseigenschaften der Kategorie *Provided* ändern sich üblicherweise nicht oder nur selten im Scope einer Anwendungssitzung. Sie werden über Austauschformate für Kompositionsfragmente spezifiziert und gelangen so in die Repositorys AppRe bzw. CoRe. Zur Laufzeit einer Mashup-Anwendung erfahren sie keine Aktualisierung. Erst nach einer Änderung der Modelle im Repository können Modifikationen durch erneute Integration bzw. Ausführung der Laufzeitumgebung bekannt gegeben werden. In der Mashup-Plattform CRUISE sind die Austauschformate für Komponenten und Anwendungen durch SMCDL und MCM spezifiziert. Sie werden durch diese Arbeit um serialisierte Formen der Qualitätseigenschaften erweitert. Bereitgestellte Eigenschaftswerte sind in Deskriptoren für Mashup-Komponenten, die in SMCDL formuliert werden, strukturell als

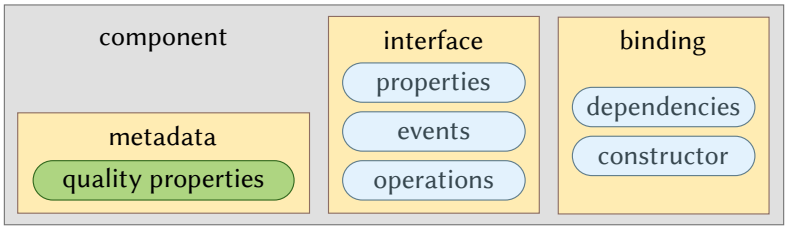

Abbildung 4.8: Integration bereitgestellter Werte für Qualitätseigenschaften im Komponentendeskriptor (SMCDL)

Metadaten eingebunden. Abbildung 4.8 zeigt diesen neuen Bestandteil in grüner Farbe und die wesentlichen bisherigen Bestandteile.

Der Beispielcode eines Instanzdokumentes der SMCDL für eine Mashup-Komponente ist in Codeausschnitt 4.1 zu sehen. Unter `metadata` sind dort die XML-Elemente `qp` mit den möglichen Attributen `type`, `concern` und `value` eingeordnet. Als Bezeichner für `type` werden die Namen der Typen von Qualitätseigenschaften der Klasse *Provided* genutzt, vgl. Metamodell in Unterabschnitt 4.3.2 und Referenzmodell in Abschnitt 4.6. Neben optionalen Angaben für `concern` wird die Wertebelegung unter `value` angegeben. Soll die Eigenschaft über einen mehrgliedrigen *Feature-Path* erfolgen, werden die `qp`-Elemente geschachtelt und die Blattelemente der kompositen Struktur enthalten die Wertebelegung, vgl. `Maintainer` → `Name`. Damit erfolgt eine Erweiterung des XML-Schemas der Metadaten für Mashup-Komponenten, die im Schema der SMCDL verwendet werden. Die Grammatik zur strukturellen Überprüfung der Deskriptorinhalte ist in Codeausschnitt A.1 definiert.

Codeausschnitt 4.1: Deskriptor (SMCDL-Instanz) einer Mashup-Komponente für die Übersicht einer Beratungssitzung mit Wertebelegungen für Qualitätseigenschaften

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <component xmlns="http://mmt.inf.tu-dresden.de/smcdl/1.15" id="mc://example.com/consultingoverview"
3           name="Consulting Overview" isUI="true">
4      <metadata xmlns="http://mmt.inf.tu-dresden.de/smcdl/1.15/metadata">
5          <qp type="Keyword" value="Consulting"/>
6          <qp type="Keyword" value="Process"/>
7          <qp type="Usability" concern="UI" value="3.8"/>
8          <qp type="Maintainer">
9              <qp type="Name" value="Andreas Rümpel"/>
10         </qp>
11         <qp type="Documentation" value="Display a consulting process structure"/>
12         <qp type="MinDimension">
13             <qp type="Width" value="150"/>
14             <qp type="Height" value="300"/>
15         </qp>
16     </metadata>
17     <interface xmlns:mcdl="http://mmt.inf.tu-dresden.de/models/mcdl.owl#">
18         <property name="consultingID" type="mcdl:hasDescription">
19             <default>acb54a2a-7e41-4989-9db4-7534eb303e87</default>
20         </property>
21         <event name="showDocument">
22             <parameter name="uri" type="mcdl:hasUri"/>
23         </event>
24         <operation name="setComment">
25             <parameter name="content" type="mcdl:hasDescription"/>
26         </operation>
27     </interface>
```

```
28      <binding bindingtype="mapping_simplewrapper">
29          <dependencies>
30              <dependency language="javascript">
31                  <url>https://example.com/consultingoverview.js</url>
32              </dependency>
33              <dependency language="css">
34                  <url>https://example.com/consultingoverview.css</url>
35              </dependency>
36          </dependencies>
37          <constructor>
38              <code>new ConsultingOverview()</code>
39          </constructor>
40      </binding>
41  </component>
```

Beim Registrieren neuer Komponentendeskriptoren in das Komponentenrepository sorgt ein *Konverter* für das Überführen der XML-Angaben in ein OWL-Modell. Dabei werden auch die **qp**-Elemente mit den Wertebelegungen von Qualitätseigenschaften eingelesen. Die komposite Struktur der **qp**-Elemente kann beibehalten werden, vgl. Metamodell in Abschnitt 4.3. Das Modell aller Komponenten im Repository ist nach dem Konvertieren die Grundlage für Änderungen und die Abfrage von Komponentenmetadaten zur Laufzeit kompositer Anwendungen. Es steht dort für Anfragen über ein REST-API bereit. Neben häufig genutzten Funktionalitäten, wie dem Registrieren, Aktualisieren oder Löschen von Komponenten können generische Anfragen über SPARQL-Querys durchgeführt werden. Dies wird durch die gemeinsame Wissensbasis über die vereinigten Komponentenmodelle des zugrundeliegenden semantischen Frameworks ermöglicht. Vor allem in Empfehlungsprozessen können so benutzerdefinierte Filter direkt auf den OWL-Modellen ausgewertet werden. Als Clients für das vom Repository angebotene API kommen neben der für den Betrieb von Mashups essenziellen Laufzeitumgebung auch Komponentenbrowser und andere Verwaltungswerkzeuge wie der Editor für Fuzzy-Sets in Frage. Diese Werkzeuge werden in Abschnitt 7.5 detailliert vorgestellt. Sowohl das Speichern von Wertebelegungen für Qualitätseigenschaften in Austauschformaten als auch im Repository kann von Komponenten auf komposite Anwendungen übertragen werden. Die hierarchische XML-Struktur wird dabei in das MCM integriert. Das Anwendungsrepository hält die importierten Modelle ebenso wie das Komponentenrepository über ein API vor, das Clients – wie beispielsweise einen Anwendungsbrowser als Werkzeug für das Durchsuchen von Anwendungsmodellen – bedient.

4.5.2 Integration gesammelter Eigenschaften

Werte für Qualitätseigenschaften der Klasse *Collected* werden sehr ähnlich zu den bereitgestellten Eigenschaftswerten behandelt. Es reicht hierbei jedoch nicht aus, die Belegung einmalig im Deskriptor des Kompositionsfragmentes anzugeben, da in diesem Fall eine Änderung nur über das Aktualisieren einer bereits registrierten Instanz erfolgen könnte. Eine zeitlich feingranulare Überwachung ist jedoch nicht notwendig, sodass die Verantwortung zur Pflege der Eigenschaftswerte nicht der Laufzeitumgebung überlassen werden muss. Die optimale Stelle zur Verwaltung der gesammelten Werte ist somit das Repository.

Wird beispielsweise die Eigenschaft *Usability* mit dem Concern UI betrachtet, so werden als Werte die gesammelten Nutzerbewertungen verwendet. Ein geeignetes Frontend, das in der Laufzeitumgebung verortet ist, ermöglicht Nutzern die Vergabe einzelner Bewertungen für

Kompositionsfragmente. Alternativ können neue Einzelwerte über Komponenten- und Anwendungsbrowser vergeben werden. Das zuständige Repository bietet ein API zur Verwaltung der Qualitätseigenschaften an, vgl. Unterabschnitt 7.4.3. Darüber werden neue Einzelwerte der aggregierten Eigenschaft hinzugefügt. Hierzu wird eine *Aggregationsvorschrift* genutzt, um aus vielen Werten eine einzelne Bewertung zu erstellen, die die Eigenschaft repräsentiert. Eine geeignete Funktion zur Aggregation numerischer Nutzerbewertungen ist das *arithmetische Mittel*. Wird der neue Wert für eine aggregierte Eigenschaft über das API zur Verwaltung hinzugefügt, so wendet die zugrundeliegende Logik die Aggregationsvorschrift an und aktualisiert den Gesamtwert im Modell des Kompositionsfragmentes. Von dort aus erfolgt die Nutzung der aktualisierten Werte analog zu den Qualitätseigenschaften der Klasse *Provided*.

Sind beispielsweise drei Nutzerbewertungen der Bewertungsskala 1 bis 5 mit dem arithmetischen Mittel 3 bereits gespeichert und es wird nun eine neue Bewertung mit dem Wert 4 hinzugefügt, so ergibt sich ein neuer aggregierter Wert von 3,25. Hierbei wird deutlich, dass es nicht ausreicht, nur den aktuellen aggregierten Wert zu speichern. Für die Aggregationsvorschrift im Beispiel muss außerdem mindestens noch die Anzahl bisheriger Bewertungen gespeichert werden, um ein neues arithmetisches Mittel zu errechnen. Alternativ können auch alle bisherigen Bewertungen gespeichert bleiben. Die Anforderungen an zusätzlich aufzuzeichnende Metadaten variiert mit der Wahl der Aggregationsvorschrift. So sind im Falle der Nutzung eines Reputationssystems – bzw. der vom Nutzer abhängigen Wichtung von Einzelwerten – weitere Kontextdaten des Nutzers zu berücksichtigen. Die hieraus entstehenden Möglichkeiten und Herausforderungen für den Schutz vor Vandalismus bzw. beim Anonymisieren von Rohdaten werden in dieser Arbeit nicht vertieft betrachtet.

Analog zum beschriebenen Beispiel erfolgt die Aktualisierung bei Werten von Qualitätseigenschaften, die nicht durch Nutzerbewertungen, sondern durch externe Messung zustande kommen oder von Experten eine Bewertung erhalten. Es gilt außerdem die Annahme, dass es im aktuellen Anwendungskontext ein bestimmtes Repository für das jeweilige Kompositionsfragment gibt. Die Indirektion über einen Verzeichnisdienst zum Auffinden von Repositorys existiert nicht. Somit entfällt auch der Aufwand zur Synchronisierung.

4.5.3 Integration gemessener Eigenschaften

Die Repositorys sind die ideale Stelle zur Verwaltung bereitgestellter und aggregierter Eigenschaften, da mehrere Laufzeitumgebungen gleichzeitig darauf zugreifen können. Die Wertebelegung ist dabei nicht abhängig von der sich in Betrieb befindlichen Anwendung. Anders verhält es sich bei gemessenen Eigenschaften. Hier wird der Wert direkt in der Laufzeitumgebung über geeignete Sensorik ermittelt. Ein Aktualisieren solcher Eigenschaftswerte im Repository ist hierbei nicht sinnvoll, da die Laufzeitbedingungen sich je nach Ausführungsumgebung unterscheiden. Der unmittelbar gemessene Wert ist hierbei entscheidend. So wird beispielsweise bei der Eigenschaft *Response Time* bezogen auf angebundene Web-Services der Wert bei jeder tatsächlich erfolgten Kommunikation aktualisiert. Da bei Anforderungen mit Bedingungen auf Qualitätseigenschaften mehrerer Klassen auch die Auswertung verteilt erfolgen muss, wird ein hybrides Auswertungsvorgehen definiert, vgl. Abschnitt 5.5. Die – für die zur Laufzeit einer Mashup-Anwendung gemessenen Werte erforderliche – Kontextsensorik muss dabei abhängig vom Typ der Eigenschaft von der Laufzeitumgebung ggf. unter Zuhilfenahme von Funktionalitäten der Implementierung von Komponenten bereitgestellt werden.

4.5.4 Aggregation von Qualitätseigenschaften auf Anwendungsebene

Kommen für eine zu erfüllende Aufgabe viele Kompositionsfragmente – insbesondere vorkomponierte Anwendungen – in Frage, so ist das Vergleichen und Filtern von Kandidaten besonders herausfordernd, wenn nutzerdefinierte Qualitätsanforderungen berücksichtigt werden sollen. Das situationsgerechte Spezifizieren solcher Anforderungen auf Anwendungsebene muss sich dabei auf konkrete Wertebelegungen stützen, die stark von den Beiträgen der einzelnen Kompositionsfragmente bestimmt sind. Insbesondere ist für den Nutzer interessant, wie sich Werte der Qualitätseigenschaften einzelner Komponenten im Kontext einer Anwendung bemerkbar machen. Die folgenden Beispiele zeigen, dass die Aggregation von Basiswerten dabei sehr unterschiedlich gestaltet sein kann. Sollen in einer Anwendung *alle Komponente eine Dienstantwortzeit von unter 200 ms* haben, ist eine andere Aggregation erforderlich, als wenn der Nutzer wünscht, dass *alle Komponenten einen bestimmten Autor* haben. Im ersten Fall ist beispielsweise das *Minimum* eine passende Aggregationsfunktion. Sind die externen Dienstaufrufe voneinander abhängig, kann jedoch auch die *Summe* als sinnvolle Aggregation genutzt werden. In der zweiten Anforderung ist die *Konkatenation* mit Duplikateliminierung geeignet, um die Autorennamen der Anwendungsbestandteile kompakt darzustellen. Hierzu bedarf es einer Unterstützung auf Werkzeug- und UI-Ebene sowie algorithmisch auf konzeptioneller Ebene.

Das Konzept zur Aggregation von Werten für Qualitätseigenschaften auf Anwendungsebene soll dabei Bedürfnisse des Nutzers der Entwicklungsplattform erfüllen. Dieser möchte sich einen Überblick aller im angegebenen Repository registrierten Anwendungen verschaffen. Bei der Auswahl einer Anwendung sollen dann aggregierte Qualitätseigenschaftswerte der enthaltenen Komponenten übersichtlich angezeigt werden, sodass ein passgenauer erster Eindruck der Gesamtanwendung vermittelt wird. Sobald die den anpassbaren Qualitätsanforderungen entsprechende Anwendung gefunden wurde, soll diese im Sinne der Prozessintegration möglichst nahtlos in der MRE gestartet und ausgeführt werden können.

Darüber hinaus ergeben sich konzeptionelle und algorithmische Anforderungen, die die Modellierung insgesamt und die Integration in die Mashup-Infrastruktur betreffen. Einer grundlegenden Übersicht und ggf. Sortierung schließt sich meist eine differenzierte Suche mit nutzerspezifischen Anforderungen an. Diese Aufgabe übernimmt der Anforderungsassistent, der die volle Mächtigkeit des Konzeptes der hier genutzten Qualitätsanforderungen beherrscht, siehe Abschnitt 5.4. Außerdem sollen im Nutzerprofil Vorgabeeinstellungen zur Konfiguration beliebter Aggregationsvorschriften abgelegt werden können. Im Sinne einer optimalen Performanz gilt es, die Anfragen an die genutzten Modelle für Qualitätseigenschaften zu optimieren und Redundanzen zu vermeiden. Sowohl der Verarbeitungsprozess als auch das Werkzeug zum Browsen von Anwendungen mit den aggregierten Eigenschaftswerten berücksichtigen die Erweiterbarkeit des Referenzmodells der Qualitätseigenschaften, damit keine grundlegenden Änderungen beim Hinzufügen oder Entfernen von Eigenschaften bzw. von Aggregationsvorschriften erforderlich sind.

Die Integration der am Aggregationskonzept beteiligten Infrastruktur wird in Abbildung 4.9 gezeigt. Daraus geht hervor, dass neben den Qualitätseigenschaften – hier vereinfacht für Metamodell und Referenzmodell stellvertretend – auch die Aggregationsvorschriften modellhaft beschrieben sind. Neben den Vorschriften in Form von Funktionen beinhaltet das Modell auch die Zuordnung zu bestimmten Eigenschaftstypen. Die Modelle werden im Anwendungsbrowser dafür genutzt, um Basiswerte und aggregierte Werte zu visualisieren. Die Belegungen der

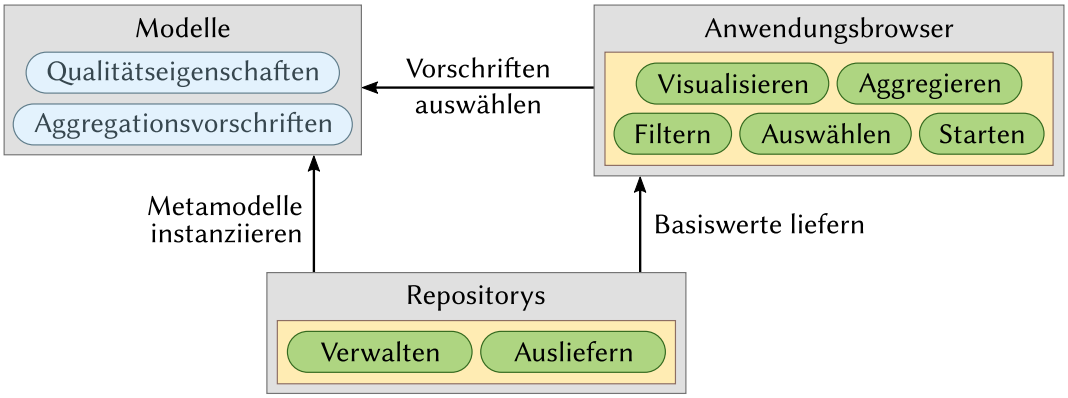

Abbildung 4.9: Eigenschaftsaggregation auf Anwendungsebene mit ausgewählten Bestandteilen der Architektur für komposite Web-Mashups

Basiswerte liefert das Komponentenrepository. Das Anwendungsrepository bietet den Zugriff auf alle verfügbaren Mashup-Anwendungen, die im Werkzeug begutachtet werden können. In der Abbildung sind diese beiden Repositorys gemeinsam in einem Kasten unten dargestellt.

Das Vorgehen bei der Aggregation von Qualitätseigenschaften auf Anwendungsebene ist in Abbildung 4.10 zu sehen. Jenseits der zwischengespeicherten Verarbeitung der Daten beginnt der Aggregationsprozess mit der Auswahl einer Anwendung im Anwendungsbrowser. Zunächst wird das Aggregationsmodell geladen, das die Zuordnung von Aggregationsvorschriften zu den Qualitätseigenschaftstypen enthält. Die daraus gewonnene Vorgabezuordnung kann durch das ebenfalls an dieser Stelle geladene Nutzerkontextmodell überschrieben werden. Aus dem bekannten Anwendungsmodell bzw. MCM kann nun die Liste aller enthaltenen Komponenten extrahiert werden. Sie ist im *Conceptual Model* enthalten. Anschließend werden über jedes Komponentenmodell die Wertebelegungen bezogen. Mit der über das Werkzeug ausgewählten bzw. aus dem Modell vorgegebenen Aggregationsfunktion können nun zu jeder Qualitätseigenschaft die aggregierten Werte ermittelt werden. Diese werden schließlich dem Nutzer als Anwendungsdetails angezeigt. Er hat nun die Möglichkeit, die ausgewählte bzw. empfohlene Funktion – und damit die Aggregationsvorschrift – für jede Qualitätseigenschaft zu ändern, eine andere Anwendung auszuwählen bzw. die Anwendung in der Laufzeitumgebung direkt zu starten.

Aggregationsfunktionen sind eine kompakt zu beschreibende Form von Aggregationsvorschriften. Die Eignung bestimmter Arten von Funktionen unterscheidet sich nach dem Datentyp der Qualitätseigenschaften. In Tabelle 4.3 werden einfache Funktionen den Datentypen für Qualitätseigenschaften zugeordnet. Bestimmte Funktionen sind unabhängig vom Datentyp anwendbar. Dazu zählen grundsätzlich alle Funktionen, die einen Wert der Eingabemenge als Aggregationswert auswählen, siehe mittlere Spalte. Dabei sind neben dem häufigsten Wert weitere Funktionen in Klammern angegeben, da Funktionen wie das Minimum zwar für sortierte Strings theoretisch anwendbar sind, der praktische Nutzen jedoch begrenzt ist. Es wird außerdem angenommen, dass die Einzelwerte nicht in einer bestimmten Reihenfolge geliefert werden. Somit werden keine Funktionen wie *erster Wert* berücksichtigt. In der rechten Spalte sind generierende Funktionen angegeben. Dort ist der Funktionswert nicht zwingend Element der Menge der Eingabewerte bzw. er gehört nicht demselben Datentyp an. So ist etwa der Anteil der `true`-Werte

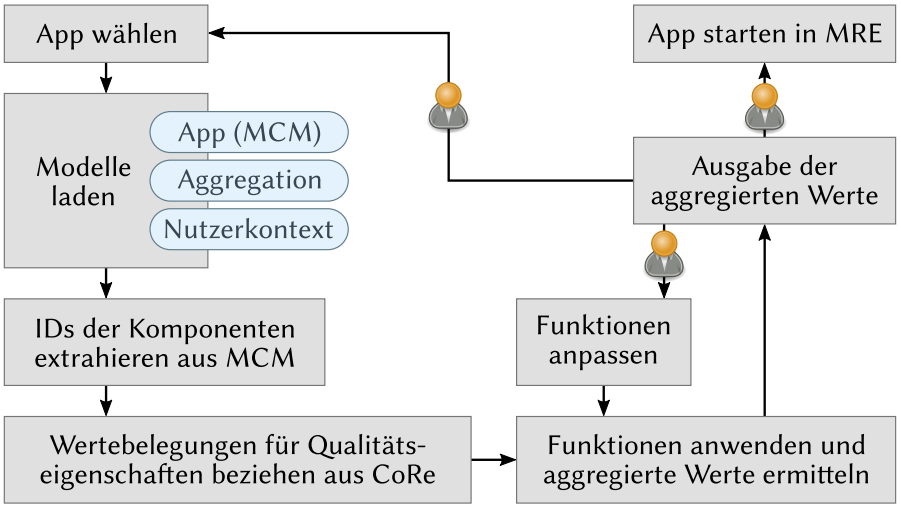

Abbildung 4.10: Prozess der Aggregation von Qualitätseigenschaften

Tabelle 4.3: Typische Funktionen für die Aggregation von Werten für Qualitätseigenschaften abhängig von deren Datentypen

Datentyp	**Auswählende Funktionen**	**Erzeugende Funktionen**
alle	Häufigster Wert (auch Min, Max und Median)	Anzahl
Boolesch	Min und Max	Anteil der `true`-Werte
numerisch	Min, Max und Median	Summe, Durchschnitt, gewichtete Funktionen
String	–	Konkatenation

selbst kein Boolean-Wert, sondern eine gebrochene Zahl zwischen 0 und 1. Neben einer einfachen Konkatenation bei Strings ist auch das Eliminieren von Dopplungen und das Einfügen von Trennzeichen als erweiterte Funktion sinnvoll, beispielsweise beim aggregierten Betrachten von Keywords, die in einer Anwendung durch die Komponenten beigesteuert werden. Für spezielle Anwendungsfälle sind grundsätzlich auch komplexere Aggregationsfunktionen möglich, vgl. *Ordered Weighted Averaging Operations* in [Yag88]. Anpassungen von einfachen Grundfunktionen aus der Tabelle sind außerdem über Charakteristika der Komposition und des Nutzerkontextes angebracht. Auf Basis von Rollen und Mustern, die sich auf den Anwendungsbestandteilen bilden lassen, ist die gewichtete Wertaggregation möglich. Über die Komponentenrollen *Master*, *Slave* und *Filter* aus [Cap+10] kann so die Relevanz jeder Komponente für die Anwendung abgeschätzt werden. Neben der automatischen Zuordnung von Rolle zu Gewicht erfolgt alternativ auch eine manuelle Konfiguration, beispielsweise über die Werkzeugunterstützung im Anwendungsbrowser.

Die beschriebene Zuordnung zwischen Qualitätseigenschaften aus dem später vorgestellten Referenzmodell, deren Datentypen und den anwendbaren bzw. empfohlenen Aggregationsfunktionen ist im Aggregationsmodell in Codeausschnitt A.4 festgelegt. Abbildung 4.1 zeigt die Integration des Modells und die Abhängigkeiten zu den anderen Modellen der Mashup-Plattform. Aggregierte Eigenschaftswerte im Anwendungsbrowser erweitern den iterativen Suchprozess

Tabelle 4.4: Vorgabewerte für Qualitätseigenschaften nach Datentyp

Wert	Bedeutung	Datentyp und Anwendung
`0`	numerisches Null	numerische Werte wie Ratings und Anteile, bspw. für Gebrauchstauglichkeit
∞	numerisches Unendlich	numerische Werte als $+\infty$ und $-\infty$, bspw. für Antwortzeit
`true`	boolesches Wahr	boolesche Werte
`false`	boolesches Falsch	boolesche Werte
`{}`	leere Menge	nicht belegte Menge bei Metadaten, bspw. für Keywords
`""`	leere Zeichenkette	nicht angegebene Metadaten, bspw. für Entwicklername

nach einer geeigneten kompositen Mashup-Anwendung für eine bestimmte Aufgabe. Dieser erweiterte Prozess wird in Abschnitt 6.1 vorgestellt. Den Funktionsumfang des Aggregationsbrowsers, der die hier genutzte Aggregationsinfrastruktur nutzt, beschreibt Unterabschnitt 7.5.3.

4.6 Referenzmodell der für Mashups typischen Qualitätseigenschaften

Die in den untersuchten praktischen und Forschungsansätzen genutzten Qualitätseigenschaften unterscheiden sich stark je nach Art des betrachteten Typs der kompositen Anwendung oder des Systems. So spielen bei Web-Services Eigenschaften mit Bezug zum UI keine Rolle. Ansätze für Web-Anwendungen mit Usability-Fokus betrachten dagegen kaum Belange der Anbindung von Services. Dieser Abschnitt fasst daher typische Qualitätseigenschaften für komposite Web-Mashups in einem Referenzmodell zusammen. Das Modell konzentriert sich dabei auf die in Unterabschnitt 2.2.2 umrissenen Szenarien. Von den in den Anforderungen an das Qualitätseigenschaftsmodell angeführten möglichen *Trägern* von Eigenschaften in einer Mashup-Infrastruktur – vgl. Abschnitt 4.2 – liegt der Fokus hier insbesondere auf den Mashup-Komponenten und den aus ihnen zusammengesetzten Mashup-Anwendungen. Viele dieser Eigenschaften sind für beide Träger relevant. Die Semantik und damit auch die Art und Weise der Bestimmung der Eigenschaftswerte kann sich allerdings unterscheiden. Auf diesen Sachverhalt wird beim Vorstellen der Eigenschaft jeweils eingegangen.

Neben der Zuordnung der Wertermittlung werden für jede Eigenschaft die Ausprägungen der *Wertebereiche*, *Datentypen* und *Dimensionen* spezifiziert. Sofern die Qualitätseigenschaft eine physikalische Größe abbildet, wird die Dimension angegeben. Die Eigenschaft *Response Time* stellt beispielsweise eine Zeitdauer dar und kann folglich in einer passenden Einheit dargestellt werden. Die kohärente Einheit nach dem internationalen Einheitensystem ist hier die Sekunde. Bestimmte Eigenschaften sind *Größen der Dimension Zahl* – veraltet auch: *dimensionslose Größen*, vgl. [ISO80000]. Da hier statt physikalischen Größen *Anzahlen* – beispielsweise Datenmengen in Bit – oder *Anteile* in Prozent bzw. in Form von Bewertungssternen (englisch: *five star rating*) abgebildet werden sollen, besitzt die Qualitätseigenschaft stattdessen nähere Angaben zum *Datentyp*, der die Struktur und die Wertebereiche der Wertebelegungen beschreibt. Zusätzlich geben Vorgabe- bzw. *Defaultwerte* die Flexibilität, Kandidaten vergleichen zu können, obwohl möglicherweise einzelne Werte nicht belegt, vergeben oder gemessen wurden. Tabelle 4.4 bietet eine Übersicht möglicher Vorgabewerte für Qualitätseigenschaften.

Um Anforderungen mit den in Abschnitt 4.4 vorgestellten Fuzzy-Sets umzusetzen, werden zu den Eigenschaften jeweils einige passende Terme angegeben. Diese können neben der als Vorgabe modellierten Funktion auch auf die Benutzer oder Anwendungsdomäne zugeschnittene Funktionen enthalten. Die Strukturierung des Modells erfolgt unter Berücksichtigung des auf Aktivitäten basierenden Ansatzes ABQM zur Erstellung von Qualitätsmodellen, siehe Abschnitt 3.2. Teile eines Systems im Sinne von ABQM werden im hier vorgestellten Modell als Concerns interpretiert, da sie definieren, auf welchen konkreten Concern einer Mashup-Komponente bzw. -anwendung sich eine bestimmte Qualitätseigenschaft bezieht. Dabei wird nicht eingeschränkt, ob eine Eigenschaft zur Laufzeit oder zur Entwicklungszeit der Mashup-Anwendung Gültigkeit besitzt. Sie kann diesbezüglich universell, etwa innerhalb von Qualitätsanforderungen, genutzt werden. Die Einflussnahme der Ausprägung einer Teileigenschaft auf eine übergeordnete Eigenschaft wird anlehnend an die Methode FCM realisiert, siehe Abschnitt 3.2. Dabei wird keine starre Zuordnung zu Ebenen in einer Hierarchie, sondern eine bedarfsweise Gruppierung vorgenommen. Durch den Einsatz von Empfehlungssystemen auch während des Prozesses der Anforderungsdefinition wird keine explorative Suche von Eigenschaften nach ihrem Typ angestrebt. Eigenschaften, die als Teile einer übergeordneten Eigenschaft bewertet werden, sind mit einem Pfeil (←) und der Nennung der übergeordneten Eigenschaft gekennzeichnet. Teilweise werden Eigenschaften nur andeutungsweise als komposit beschrieben. Die Dekomposition ist dann Teil einer möglichen Erweiterung des Referenzmodells.

Es werden zunächst die Qualitätseigenschaften vorgestellt, die sich *Concerns* zuordnen lassen. Anschließend folgen die Eigenschaften, die Komponenten oder Anwendungen im Ganzen beschreiben. Das hiermit vorgestellte Referenzmodell enthält Qualitätseigenschaften, die typisch für das Anwendungssystem Web-Mashup sind. Zum Vergleich ist weiter unten angegeben, ob und wie häufig die jeweilige Eigenschaft in der Literatur bereits bekannt ist und ob sie in ihrer Semantik verändert wurde. Die Auswahl der Eigenschaften wurde mit besonderem Blick auf die Eignung innerhalb der Anwendungsszenarien dieser Arbeit vorgenommen. Es wird jeweils zu den im Metamodell beschriebenen strukturellen Eigenschaften eine Aussage getroffen.

4.6.1 Eigenschaften mit Concerns

Folgende – für komposite Web-Mashups typische – Qualitätseigenschaften können einem oder mehreren Concerns, vgl. Unterabschnitt 4.3.1, zugeordnet werden. Die englischen Begriffe sind bei Bedarf in Klammern angegeben. Bei den Metriken wird teilweise auch darauf eingegangen, wie die Wertebelegung von Komponenten in einer Anwendungsmetrik aggregiert wird. Die Auflistung in diesem Kapitel enthält nur einen kleinen Ausschnitt aus dem Referenzmodell für Qualitätseigenschaften, indem die für die Beispielszenarien in Unterabschnitt 2.2.2 relevanten Eigenschaften genannt werden. Die restlichen Eigenschaften sind in Anhang B beschrieben.

QE Aktualität (Currentness)

Wie aktuell sind die Daten, die auf der Benutzeroberfläche präsentiert werden? Es wird außer Acht gelassen, wie diese Daten möglicherweise von Hintergrunddiensten nachgeliefert werden, da es sich um Black-Box-Komponenten handelt. Ob eine automatische Aktualisierung vorliegt und somit eine direkte Abhängigkeit besteht, wird durch die Zugriffsart der Komponenten angegeben. Wenn die Aktualisierung beispielsweise nur durch Anfragen des Nutzers ausgelöst

wird, kann als Wert für die Aktualität 0 angegeben werden.

- **Ermittlung:** statisch, **Dimension:** Zugriffe pro Zeiteinheit (s^{-1}), **Vorgabewert:** 0
- **Concerns:** UI-Daten (indirekt: Hintergrunddaten)
- **Terme:** veraltet, wenig aktuell, recht aktuell, möglichst aktuell, hochaktuell
- **Komponentenmetrik:** Der Wert wird vom Komponentenentwickler zur Entwicklungszeit festgelegt und entsprechend statisch angegeben.
- **Anwendungsmetrik:** Bei der Anwendung wird der Durchschnitt über alle Werte der Komponenten berechnet. n steht dabei für die Anzahl der Komponenten.

$$\text{Aktualität}_{\text{Anwendung}} = \frac{\sum_{i=1}^{n} \text{Aktualität}_i}{n}$$

QE Zufriedenheit (Satisfaction)

Wie zufrieden sind Nutzer und Entwickler mit den einzelnen Concerns der Anwendung oder der Komponente? Diese Eigenschaft ist mit der Gebrauchstauglichkeit verwandt und kann bei Bedarf in einem angepassten Modell dieser zugeordnet werden.

- **Ermittlung:** gesammelt, **Datentyp:** Float (Five-Star-Rating), **Vorgabewert:** 0
- **Concerns:** Hintergrunddienste, Funktionalität, UI-Daten, Hintergrunddaten, UI, API bei der Komponentenebene
- **Terme:** niedrig, mittel, möglichst hoch, hoch
- **Komponenten- und Anwendungsmetrik:** Es wird die Bewertung von Komponente oder Anwendung durch Nutzer und Entwickler (nur bei den Komponenten) angegeben, wobei die Bewertung zu den Daten im Hintergrund und des Komponenten-API nur von Entwicklern gemacht werden sollten, da Nutzer über diese in der Regel nicht genügend Informationen erhalten. Es wird jeweils die Zufriedenheit zu einem einzelnen Concern bewertet und der Durchschnitt der Bewertungen angegeben. Zusätzlich kann auch ein Durchschnittswert der Concernbewertungen als Wert für die Anwendung bzw. Komponente angegeben werden.

QE Antwortzeit (Response Time)

Zeit, die der Hintergrunddienst bzw. das UI der Komponenten benötigt, um auf eine Netzwerkanfrage zu antworten. Die UI-Antwortzeit wird indirekt von der Funktionalität beeinflusst, für die Funktionalität wird also kein separater Wert angegeben.

- **Ermittlung:** dynamisch und gesammelt, **Dimension:** Zeit (s), **Vorgabewert:** ∞
- **Concerns:** Hintergrunddienste, UI (indirekt: Funktionalität)
- **Terme:** hoch, mittel, möglichst niedrig, niedrig

- **Komponentenmetrik:** Die Messung erfolgt durch einen Kontextmonitor bei jeder ausgeführten Netzwerkanfrage. Es werden also separate Messungen für beide Concerns vorgenommen. Als Eigenschaftswert wird dann der jeweils letzte ermittelte Wert angegeben.
- **Anwendungsmetrik:** Für die Anwendung wird der Durchschnitt von allen Werten der enthaltenen Komponenten angegeben:

$$\text{Antwortzeit}_{\text{Anwendung}} = \frac{\sum_{i=1}^{n} \text{Antwortzeit}_i}{n}$$

Tabelle 4.5 zeigt eine Zusammenfassung von Eigenschaften mit Concern-Zuordnung. Sie ist eine Weiterentwicklung der Vorarbeit zu den Qualitätseigenschaften aus [Rüm+13]. Dabei steht UID für UI-Daten, BGD für Hintergrunddaten, FCT für Funktionalität und SVC für Hintergrunddienste. Die Qualitätseigenschaften sind dabei nach der Art ihrer Wertermittlung gruppiert. Um die Zuordnung zu den Konzepten aus den Arbeiten des Standes der Forschung und Technik, die jeweils hinter der Eigenschaft angegeben sind, zu erleichtern, werden die englischsprachigen Eigenschaftsnamen angegeben. Die Einrückung in der Tabelle gibt die bereits vorgenommene Strukturierung der vorgestellten Eigenschaften wieder. Aus der letzten Spalte wird auch deutlich, welche Eigenschaftswerte von Komponenten auf Anwendungsebene aggregierbar sind und welche nur auf Anwendungsebene existieren, wie etwa *Component Suitability*. In der Auflistung der Eigenschaften wird neben der *messbaren* Größe *Datenverkehr* auch die *statisch vorgegebene* Eigenschaft *Übertragungsrate* geführt. Dieses Eigenschaftspaar zeigt beispielhaft, dass eine Basisgröße durch zwei voneinander abgegrenzte semantische Aspekte beschrieben werden kann, die hier anhand der Wertermittlung erfolgt. Somit können verwandte oder identische Qualitätseigenschaften, die durch eine voneinander verschiedene Art der Wertermittlung zustande kommen, parallel im Modell existieren und als Hilfestellung für die Formulierung von Qualitätsanforderungen genutzt werden.

4.6.2 Eigenschaften ohne bestimmte Concerns

Neben den Eigenschaften, denen sich verschiedene der in Unterabschnitt 4.3.1 vorgestellten Concerns zuordnen lassen, existieren auch solche, die sich primär auf die gesamte Mashup-Komponente oder -Anwendung beziehen oder sie im Allgemeinen beschreiben. Dazu zählen auch klassische *Metadaten*, wie der Name des Entwicklers einer Komponente, die sich auf die gleiche Art und Weise wie die bereits vorgestellten Eigenschaften in die anforderungsgetriebenen Entwicklungs- und Nutzungsprozesse integrieren lassen. Die Beschreibungen enthalten teilweise auch Alternativen, wie beispielsweise minimale und maximale Dimensionswerte. Diese zerfallen in der Implementierung des Referenzmodells in separate Eigenschaften bzw. sind hierarchisch aufgebaut und können über einen *Feature-Path* adressiert werden. Exemplarisch sind hier die Eigenschaften aufgelistet, die in den Beispielszenarien auftreten. Weitere Qualitätseigenschaften sind wie in der anderen Gruppe in Anhang B beschrieben.

QE Preis (Price)

Wie viel kostet die Nutzung der Komponente oder Anwendung? Der Preis ist einer bestimmten *Währung* – standardmäßig EUR – zugeordnet. Der Preis der Anwendung ist die Summe der Preise der Komponenten.

Tabelle 4.5: Qualitätseigenschaften mit Zuordnung bestimmter Concerns

Bereitgestellt	API	UID	BGD	FCT	SVC	UI	A|C
▲ Interoperability [CDM09]	●		●				▪
▲ Data Exchange Formats [CDM09]	●		●				▪
▲ Programming Languages [CDM09]	●			○			▪
▲ Protocols [CDM09]	●						▪
♦ Actuality (Currentness, Currency) [OSM08]		●	○				▦▪
▲ Credibility [OSM08; Ore10]		●	○		○		▦▪
♦ Security [Lee+03; ISO25010; OSM08; Ore10; Pap+06; Ran03]			●	●			▦▪
▲ Authentication [Ran03; Lee+03]				●			▦▪
▲ Authorization [Ran03; Lee+03]				●			▦▪
▼ Data Storage			●				▦▪
▲ Data Encryption [Ran03; Lee+03]			●				▦▪
▲ Traceability [Ran03; Lee+03]				●			▦▪
▼ Style						●	▦▪
▲ Familiarity [Ore10]						●	▦▪
▼ Source					●		▦▪
Gesammelt	API	UID	BGD	FCT	SVC	UI	A|C
▲ Component Suitability [Cap+11a]	●	●	●	●	●	●	▦
♦ Consistency [MRW77; Ore10; GC87; Cap+11a]		●		●		●	▦
▲ Adaptability [ISO25010; ISO9126; MRW77; Ore10; GC87]		●		●		●	▦▪
▲ Usability [Ore10; MRW77; GC87; ISO25010; Cap+11a; CDM09]	◗					●	▦▪
▲ Learnability [Ore10; ISO25010; ISO9126; CDM09]	◗					●	▦▪
▲ Accuracy [OSM08; Ore10]		●					▦▪
♦ Completeness [Ran03; MRW77; OSM08; ISO25010]	◗	●	●	●		●	▦▪
▲ Satisfaction [Ore10]	◗	●	●	●	●	●	▦▪
▲ Accessibility [OSM08; Ore10]		○				●	▦▪
Gemessen	API	UID	BGD	FCT	SVC	UI	A|C
▼ Data Traffic			●				▦▪
♦ Reliability [ISO25010; Ore10; Pap+06; Ran03; Lee+03; GC87]		○	○	●	●	●	▦▪
♦ Availability [ISO25010; Ore10; Pap+06; Ran03; Lee+03; OSM08]		○	○	●	●	●	▦▪
♦ Robustness [Ran03; Lee+03]		○	○	●	●	●	▦▪
▲ Response Time [ISO25010; Ore10; Pap+06; Ran03; Lee+03; GC87]				○	●	●	▦▪
♦ Timeliness [CDM09; Cap+11a]				○	●	●	▦▪
▼ Deadline				○	●	●	▪

Symbolerklärung

▲	aus existierendem Modell	●	Concern direkt anwendbar
♦	mit veränderter Semantik	○	Concern indirekt betroffen
▼	neue Eigenschaft	◗	Concern nur für Komponenten anwendbar
▦	auf Anwendungen zutreffend	▪	auf Komponenten zutreffend

- **Ermittlung:** statisch, **Datentyp:** Float, **Vorgabewert:** ∞
- **Terme:** teuer, möglichst günstig, günstig, kostenlos

QE Energieverbrauch (Energy Consumption)

Wie viel Energie verbraucht die Komponente oder Anwendung durchschnittlich während der Laufzeit?

- **Ermittlung:** statisch, **Dimension:** Leistung (Watt), **Vorgabewert:** ∞
- **Komponenten- und Anwendungsmetrik:** Es wird der durchschnittliche Verbrauch pro Zeit angegeben. Es wird dann der Durchschnittswert über alle gemessenen Werte angegeben.

4.7 Zusammenfassung und Bewertung des Modells für Qualitätseigenschaften

Ziel der Modellierung von Qualitätseigenschaften ist das einheitliche Bereitstellen von Basisgrößen für Qualitätsanforderungen, die durch Messung oder auf andere Art und Weise zustande kommen. Technisch sollen diese Eigenschaften und ihre Wertebelegung sowohl durch Sensorik bzw. Verwaltungseinheiten befüllt werden können als auch für Qualitätsanforderungen referenzierbar sein. Durch die Formulierung in RDF und die Trennung von Metamodell und Instanzen über OWL wurde die Grundlage für die Integration und Zusammenarbeit der Modellierungslandschaft in der Mashup-Plattform geschaffen, die auch die Qualitätsanforderungen und weitere Domänenmodelle sowie Kontextmodelle zur Speicherung umfasst. Außerdem wird dadurch die Anforderung der Erweiterbarkeit des Modells umgesetzt. Dies gilt sowohl für strukturelle Besonderheiten als auch für das Hinzufügen und Ändern von Eigenschaften, die vom Referenzmodell abweichen. Weiterer Vorteil dieser Art der Modellierung ist das Bilden von Synergien mit existierenden Drittmodellen, beispielsweise für das Anzeigen von Einheiten und ggf. die automatische Konvertierung zwischen Einheitensystemen mit Hilfe von QUDT.

Die durch die ISO-Norm abstrakt zur Gruppierung eingeführte und auch bei FCM vorherrschende *hierarchische Strukturierung* von Qualitätseigenschaften ist in flacher Form auch in dem hier vorgeschlagenen Qualitätsmodell umgesetzt. Bestimmte Qualitätseigenschaften aus dem Referenzmodell, wie beispielsweise *Interoperabilität*, setzen dieses Prinzip um. Hauptvorteil hiervon ist das Bilden von aggregierten Werten und dadurch eine verbesserte Erfassbarkeit für die Nutzer der Mashup-Plattform. Eine tiefere Hierarchie ist für das typischerweise nicht explorative Festlegen bei der iterativen Empfehlung für den Nutzer nicht sinnvoll. Neben dem hierarchischen Anordnen von Eigenschaften in einem Modell und den damit verbundenen Vor- und Nachteilen ist auch das Facettieren von Eigenschaften nach bestimmten *Belangen* von Kompositionsfragmenten sinnvoll zur Strukturierung. Auf Grundlage dieses Gedankens können den Eigenschaften Concerns zugeordnet werden. Das Referenzmodell für diese Concerns spiegelt die wichtigsten Belange von Mashup-Bestandteilen wider, vgl. Unterabschnitt 4.3.1.

Ein essenzielles Konzept zur Gruppierung von Qualitätseigenschaften anhand des Zustandekommens von Eigenschaftswerten ist die Typisierung nach dem Modus der Wertermittlung. Dabei wurden drei Typen identifiziert: *Statisch festgelegt* (über das Komponentenmodell als Metadatum), *Gemessen* (über die Sensorik der Laufzeitumgebung) und *Gesammelt* (über andere Eigenschaften als Basiswerte mit einer Aggregationsinfrastruktur). Diese Aufteilung ist sehr wichtig zur Verortung von Aufgaben der Verwaltung innerhalb der Mashup-Plattform und für das konkrete Vorgehen während der Auswertung von Qualitätsanforderungen, die ggf. verteilt erfolgt. Wurde die gewünschte Qualitätseigenschaft ausgewählt, stellt das adäquate Festlegen von Vergleichswerten eine Herausforderung – vor allem für unerfahrene Nutzer – dar. Unscharfe Anforderungen mit *Fuzzy-Sets*, die auf Qualitätseigenschaften definiert werden, bieten in Verbindung mit *Fuzzy-Termen* einen erleichterten Zugang für die Nutzung von Empfehlungsfunktionen während des nutzergetriebenen Erstellens von Anforderungen, vgl. These 3 in Unterabschnitt 1.2.2. Bestimmte Terme müssen für die Vergleichbarkeit, die für Funktionen im später vorgestellten Anforderungsassistenten benötigt wird, in ihrem Wertebereich limitiert werden. Dies steht in Konkurrenz zu grundsätzlich offenen Datentypen, die beispielsweise bis ins Unendliche numerische Werte vorsehen und in Vorgabewerten genutzt werden können. In

Kombination mit offenen Fuzzy-Sets, wie beispielsweise *möglichst gering*, ist pro Eigenschaft das Setzen von *Limits* als charakteristische untere und obere Begrenzung erforderlich. Diese werden auch als Hilfsmittel für den Editor für Fuzzy-Sets zur Begrenzung des sichtbaren Bereichs (englisch: *viewport*) des Funktionsgraphen genutzt, vgl. Unterabschnitt 7.5.2.

Das Referenzmodell hinterlegt die Szenarien mit konkreten Eigenschaftstypen und grenzt zugleich die typische Ausprägung von Qualitätseigenschaften in einer Mashup-Plattform von Ansätzen anderer Domänen und den Vorschlägen allgemeinerer Qualitätsmodelle ab. Eine Teilmenge dieser Eigenschaften aus dem Referenzmodell kommt insbesondere in den Anwendungsszenarien dieser Arbeit zum Einsatz, vgl. Unterabschnitt 2.2.2. Sie erfahren dadurch eine zusätzliche Validierung innerhalb ihres Einsatzes in Beispielanforderungen, sowie durch ihre Verwendung als Beispieldaten in der Implementierung. Weitere Eigenschaften werden in Anhang B beschrieben. Ein Überblick zur Verbreitung der Qualitätseigenschaften ist in Tabelle 4.5 zu sehen sowie in den einzelnen Abschnitten im Kapitel zum Stand der Forschung und Technik zur jeweiligen Forschungsarbeit aufgeführt. Nutzern der in dieser Arbeit vorgestellten Mashup-Plattform steht es frei, eine geeignete Teilmenge der Referenzeigenschaften für ihren Anwendungsfall auszuwählen bzw. das Modell zu ergänzen.

Existierende Modelle beinhalten bestimmte Qualitätseigenschaften, die sich innerhalb der hier genutzten Mashup-Plattform als Qualitätsanforderungen erübrigen, da sie bereits durch andere Systeme in der Laufzeitinfrastruktur berücksichtigt werden. Exemplarisch stellt das Qualitätsmodell für Mashup-Komponenten nach Cappiello u. a. die *Interoperabilität* als eine Qualitätseigenschaft vor, die auch in diesem Referenzmodell geführt wird. Als zugeordnetes Konzept gelten dafür in der Plattform CRUISE die semantischen Annotationen für Datentypen. Durch sie wird in einer anderen Phase des Empfehlungsprozesses eine Interoperabilität im Sinne der *Kompatibilität von Schnittstellen und Datenformaten* – vgl. Mediation in [Rad+14] – umgesetzt. Das Festlegen der Interoperabilität als Qualitätseigenschaft im Rahmen der Anforderungsmodellierung durch Nutzer ist somit nicht erforderlich, da die Verantwortlichkeit auf die Mashup-Plattform übergeht und somit automatisiert behandelt werden kann.

Kapitel 5

Festlegen und Auswerten von Qualitätsanforderungen

Qualitätsanforderungen entstehen durch bewusstes Spezifizieren eines Benutzers oder Entwicklers, durch implizites Erzeugen aus Interaktionsmustern bzw. Kontextmodellen oder als Folge des automatisierten Aushandelns zwischen Diensten und Geräten. Über ihre charakteristischen Bedingungen manifestieren sie sich in Softwaresystemen als *Zusicherungen* oder *Erfordernisse* in unterschiedlichen Phasen der Entwicklung und Nutzung, um die *gewünschte Produktqualität* bestimmter Artefakte zu beschreiben. Die automatisierte Auswertung solcher Bedingungen baut unmittelbar auf die bereits vorgestellte Modellierung der Qualitätseigenschaften und ihrer Wertermittlung auf. Vor allem in Kombination mit funktionalen Anforderungen an die Fähigkeiten von Anwendungen und Anwendungsbestandteilen sind die Einsatzzwecke von Qualitätsanforderungen vielfältig. In einer Plattform für komposite Web-Mashups können mittels intelligenter Werkzeugunterstützung sowie der Modellierungs- und Auswertungsinfrastruktur über Empfehlungssysteme und Selbstadaption damit sowohl Entwicklungs- als auch Nutzungsprozesse maßgeblich verbessert werden. Untersuchungsgegenstand dieses Kapitels sind daher Komposition, Wichtung, Konfiguration und Herkunft von Qualitätsanforderungen in einer Mashup-Plattform.

Nachdem zuerst die spezifischen Herausforderungen, die sich bei der formalen Definition von Qualitätsanforderungen in Web-Mashups ergeben, charakterisiert werden, nimmt dieses Kapitel eine architektonische Einordnung der hier betrachteten Anforderungstypen bezüglich ihrer beteiligten Rollen bei der Entstehung und Auswertung vor. Anhand eines Metamodells wird anschließend der Aufbau von Qualitätsanforderungen unter Berücksichtigung eingebundener Konzepte aus dem vorhergehenden Kapitel und weiterer Domänenmodelle beschrieben. Basis dafür ist insbesondere das zuvor präsentierte Eigenschaftsmetamodell mit der Möglichkeit zur Angabe von Fuzzy-Termen. Zum nutzergetriebenen und generisch ausgelösten Festlegen von Anforderungen wird hier lediglich ein Überblick gegeben, da im nächsten Kapitel eine detaillierte Einordnung in den Prozess der Mashup-Entwicklung vorgenommen wird. Dies betrifft vor allem die den Anforderungen zugeordneten Adaptionsaktionen, die ebenfalls Gegenstand des nächsten Kapitels sind. Schließlich werden Auswertungsprozesse für Qualitätsanforderungen mit ihren Algorithmen betrachtet, die abhängig von den referenzierten Eigenschaftstypen in unterschiedlichen Stellen der Kompositions- und Laufzeitinfrastruktur stattfinden.

5.1 Herausforderungen im Umgang mit Anforderungen

Um *Auswertungsergebnisse* von Qualitätsanforderungen in wichtigen Prozessen der Mashup-Plattform – wie der Empfehlung von Kompositionsfragmenten – nutzen zu können, muss für

die maschinelle Verarbeitung ein *skalarer Wert* bestimmt werden. Dieser wird dann zum Sortieren einer Menge an Kandidaten oder zum Vergleich mit Schwellwerten für Adaptionsaktionen genutzt. Im einfachsten Fall existiert jedoch nur ein boolescher Wahrheitswert, der den Erfüllungsgrad einer der Qualitätsanforderung zugrundeliegenden *Bedingung* als entweder *erfüllt* oder *nicht erfüllt* anzeigt. Kontinuierliche Erfüllungsgrade dazwischen, die vor allem für Sortierfunktionen notwendig sind, werden mit der Nutzung von *Fuzzy-Sets* und *Fuzzy-Logik* ermöglicht. Das Modell für Qualitätseigenschaften wurde dafür bereits ausgelegt, siehe Abschnitt 4.4. Dieses Konzept unterstützt außerdem den Wunsch *verpflichtende* und *optionale* Bedingungen innerhalb von Qualitätsanforderungen zu verknüpfen. Das Kombinieren mehrerer gleichzeitig geltender Bedingungen geschieht durch den verknüpften Einsatz von *Junktoren*, die über die Funktionen der Fuzzy-Logik das Bilden skalarer Gesamtbewertungen ermöglichen. Auch das aus Nutzersicht vermeintliche Problem *konkurrierender* Bedingungen über verschiedene Typen von Qualitätseigenschaften wird damit behandelt. Allerdings steigt bei sehr komplex komponierten Bedingungen die intellektuelle Hürde für den Autor einer solchen Qualitätsanforderung. Soll also ein Mensch die Anforderungsquelle sein, vgl. Szenarien ① und ②, gilt es durch geeignete Werkzeugunterstützung das Festlegen von Bedingungen auf dem entsprechenden Fähigkeitsniveau zu erleichtern. Ziel dieser Werkzeuge ist das möglichst passgenaue Umsetzen der Intention menschlicher Anforderungsquellen sowohl beim Festlegen von Bedingungen als auch beim Behalten des Überblicks im Umgang mit mehreren Anforderungen und den Auswirkungen der daran gekoppelten Adaptionsaktionen. Hierbei hilft das Bilden von *Profilen* über beliebte Kombinationen von Bedingungen als *Favoriten*. Insgesamt sollen durch ein grundlegendes Metamodell für Qualitätsanforderungen alle in den Szenarien dieser Arbeit vorkommenden Anforderungsquellen und Fälle der Integration in den Entwicklungs- und Nutzungsprozess kompositer Web-Mashups abgedeckt werden. Die Instanzen des Metamodells bilden die Schnittstelle zum Autorenwerkzeug sowie zur Laufzeitplattform als Auswertungsinfrastruktur. Mit dem Modell wird eine umfassende Grundlage zum Festlegen und Auswerten anpassbarer Qualitätsanforderungen geschaffen. Als Seiteneffekt liegen auch allgemein zu optimierende Kriterien, wie etwa die Security-Ebenen aus dem Papier von Cappiello, Daniel und Matera [CDM09], im Mächtigkeitsbereich des hier vorgestellten Konzeptes.

5.2 Qualitätsanforderungen in der Mashup-Architektur

Um anpassbare Qualitätsanforderungen in der Mashup-Plattform nutzen zu können, ist eine umfassende Integration in die Architektur notwendig. Ausgangspunkt ist die Referenzinfrastruktur für komposite Web-Mashups aus Abbildung 2.3. Die wichtigsten Integrationspunkte für Qualitätsanforderungen zeigt Abbildung 5.1. Neben der Modellierung – hier die Instanzen des Metamodells für Qualitätsanforderungen als Voraussetzung für deren Speicherung – sind dies vor allem die Module zum Festlegen, Bearbeiten und Auswerten von Anforderungen.

Als zentrale Stelle der Nutzerinteraktion mit Qualitätsanforderungen erhält die Laufzeitumgebung MRE einen *Anforderungsassistenten*. Dieser ist in mehreren Modi für das initiale Festlegen und für das Bearbeiten bzw. Löschen geltender Qualitätsanforderungen verantwortlich. Dabei werden beide Fähigkeitsprofile menschlicher Akteure aus den Szenarien ① und ② unterstützt. Das heißt, dass sowohl Nutzerfreundlichkeit während der Empfehlungsphase von Kompositionsfragmenten als auch umfassende Kompositionswerkzeuge zur Überwachung von An-

forderungen zur Laufzeit angeboten werden. Bedingungen und auslösende Ereignisse sowie Adaptionsaktionen für Komponenten und Anwendungen können im Anforderungsassistenten konfiguriert werden. Sein Funktionsumfang als Werkzeug wird detailliert in Unterabschnitt 7.5.1 beschrieben.

Nicht nur die Zeitpunkte der Auswertung für Qualitätsanforderungen, sondern auch die *Auswertungsorte* unterscheiden sich je nach Anwendungsszenario. Teilweise wird eine hybride Auswertung benötigt, die mehrere *Evaluatoren* beansprucht. Diese beziehen die benötigten Vergleichsdaten beispielsweise aus der Sensorik der Laufzeitumgebung oder aus dem Komponentenrepository, das Bewertungen zu bestimmten Qualitätseigenschaften sammelt. Die Charakteristika der Anforderungsauswertung beschreibt Abschnitt 5.5. Gebunden an Nutzerprofile sollen favorisierte Qualitätsanforderungen im *Kontextdienst* als Instanzen des Anforderungsmetamodells, das in Unterabschnitt 5.3.1 beschrieben wird, gespeichert werden können. Diese Speichermöglichkeit für Qualitätsanforderungen macht sich der Anforderungsassistent zu Nutze, indem in einem neuen Anwendungskontext Anforderungen aus früheren Sitzungen zur Wiederverwendung bereitstehen.

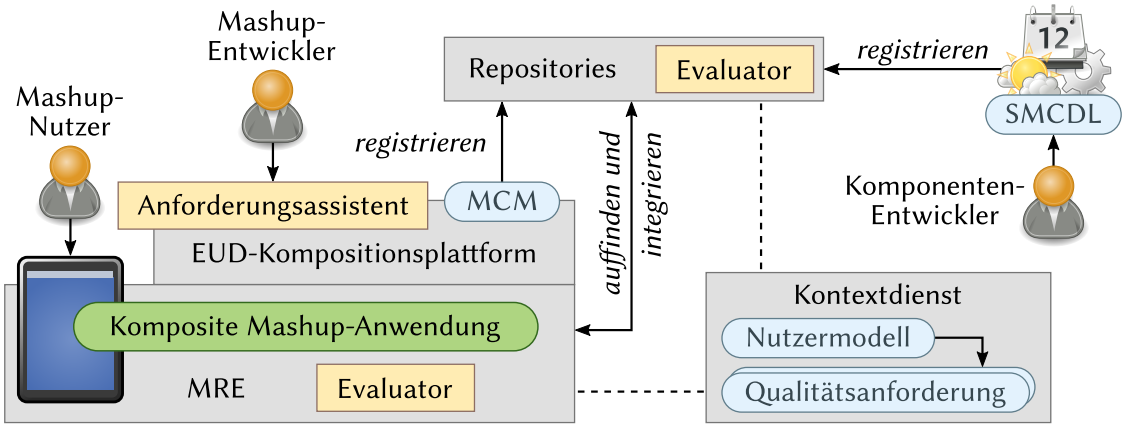

Abbildung 5.1: Einordnung der Infrastruktur zur Spezifikation und Auswertung von Qualitätsanforderungen sowie der Modelle in die Mashup-Architektur

Neben Komponenten und Anwendungen als Kompositionsfragmente in Web-Mashups gibt es weitere wichtige Stakeholder, die als Ziel von Qualitätsanforderungen in Frage kommen. Das sind insbesondere Nutzer und Entwickler als menschliche Rollen sowie das Gerät und dessen Betriebssystem bzw. Laufzeitumgebung mit der Möglichkeit des Zugriffes auf Sensorik. Sie sind potenzielle Träger von Qualitätseigenschaften, vgl. Abschnitt 4.3. Diese vier Gruppen von Zielobjekten für Qualitätsanforderungen können gleichzeitig als Anforderungsquellen auftreten. Eine Quelle ist verantwortlich für das *Stellen einer Qualitätsanforderung* bzw. für die zugrundeliegende Motivation. Aus der Menge möglicher Paarungen von Quelle und Ziel stellt Abbildung 5.2 einige Beispiele typischer Zuordnungen vor. Dabei zeigt sich, dass nicht alle Kombinationen sinnvoll bzw. typisch für den Einsatz im Entwicklungs- und Nutzungsprozess von Web-Mashups sind. Die roten und grünen Pfeile führen dabei jeweils von der Quelle zum Ziel einer Qualitätsanforderung. Rote Pfeile zeigen untypische Zuordnungen an. Bei den grünen Pfeilen sind jeweils Beispiele für mögliche Anforderungen angegeben. Typisch für Szenario ①

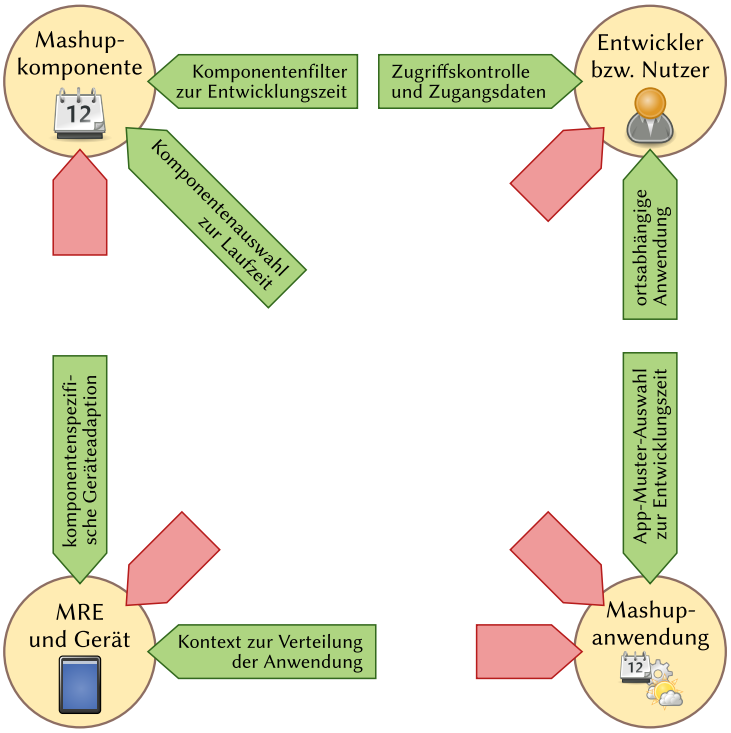

Abbildung 5.2: Typische Kombinationen der Quellen und Ziele von Qualitätsanforderungen

ist die Auswahl von Komponenten zur Entwicklungszeit auf Basis von Filtern, die dem Empfehlungssystem zur Verfügung stehen. Ohne menschliche Beteiligung stellt beispielsweise die *Komponente* Anforderungen an das ausführende *Gerät* bzw. dessen Softwareplattform.

Abhängig von der Zuordnung der Quelle und dem Ziel einer Qualitätsanforderung entsteht sie zu unterschiedlichen Zeiten und mit einer bestimmten Intention im Prozess der Entwicklung bzw. Nutzung. Die Intention bestimmt maßgeblich, wo und wie lange eine erstellte Anforderung gespeichert wird. Während der Empfehlung ist der Lebenszyklus einer Qualitätsanforderung recht kurz, da sie nur bis zur Auswahl eines empfohlenen Kompositionsfragmentes vorgehalten werden muss. Sie ist dann nur im entsprechenden Assistenten der Laufzeitumgebung gespeichert, die den Empfehlungsprozess begleitet. Anders verhält es sich mit Anforderungen, die als Favorit gespeichert werden sollen. In diesem Fall wird die Infrastruktur für den Nutzerkontext für das Speichern in Anspruch genommen, die später in Unterabschnitt 5.4.4 vorgestellt wird. Im Gegensatz dazu müssen Anforderungen, die für das Szenario ② zur Überwachung von Qualitätseigenschaften erstellt werden, mindestens so lange gespeichert werden, bis das explizit angegebene Ereignis zur Auswertung eingetreten ist. Bei zyklisch eintretenden Ereignissen muss die Anforderung sogar gespeichert bleiben, bis sie aus der Überwachung entfernt wird.

5.3 Aufbau von Qualitätsanforderungen

Die umfassende Nutzung von Qualitätseigenschaften in der Entwicklung, Auswahl und Überwachung von Web-Mashups und deren Bestandteilen erfordert eine flexible Modellierungsgrund-

lage, die dazu in der Lage ist, sich auf die Gegebenheiten der Anwendungsszenarien und deren Stakeholder einzurichten. Das *Metamodell für Qualitätsanforderungen* liefert diese Modellierungsgrundlage. Es wird als Schwerpunkt in diesem Abschnitt mit seinen strukturellen Besonderheiten vorgestellt. Aufbauend darauf wird auf die Komposition und Wichtung sowie die Operatoren bei der Verknüpfung mehrerer zusammengehörender Bedingungen innerhalb einer Anforderung eingegangen. Neben typischen Ereignissen, die die Anforderungsauswertung auslösen wird anschließend eine Vorschau auf Adaptionsaktionen gegeben, die später in einem Referenzmodell detailliert diskutiert werden. Schließlich wird dargestellt, wie weitere Domänenmodelle in das Anforderungsmodell eingebunden werden können. Anhand von Beispielen werden die strukturellen Charakteristika veranschaulicht.

5.3.1 Metamodell für Qualitätsanforderungen in Web-Mashups

Zur konsistenten Darstellung während der Speicherung und Verarbeitung bedienen sich Qualitätsanforderungen eines Metamodells, das nun im Detail vorgestellt wird. Dabei gibt es mehrere Anknüpfungspunkte an das Metamodell für Qualitätseigenschaften, das in Abschnitt 4.3 beschrieben wird. Das in Abbildung 5.3 gezeigte Metamodell für Qualitätsanforderungen nutzt ebenfalls die Visual Notation for OWL Ontologies (VOWL). Es baut auf den vorläufigen Überlegungen des in [Rüm+14] veröffentlichten Modells auf.

Eine Qualitätsanforderung (englisch: *quality requirement*) setzt das Muster Event Condition Action (ECA) um, das vor allem in regelbasierten Systemen und in der Event-Driven Architecture (EDA) eingesetzt wird. Die Charakteristika der Anforderung sind durch die entsprechenden Klassen *Event*, *Condition* und *Action* im Metamodell vertreten. Diese Events (deutsch: Ereignisse) bestimmen, welche Gelegenheiten zur Auswertung der Qualitätsanforderung führen. Dabei ist zu unterscheiden, ob die Auswertung zeitgesteuert, etwa in einem vorbestimmten Intervall, einmalig beim Erzeugen der Anforderung oder basierend auf Nutzungs- bzw. Entwicklungsaktionen in der Anwendung oder in der Laufzeitplattform ausgelöst werden soll. Zur Laufzeit einer Anwendung kann beispielsweise auch die Aktivität an Kommunikationskanälen als auslösendes Ereignis genutzt werden. Verschiedene Arten solcher Events werden durch ein Referenzmodell in Unterabschnitt 5.3.4 beschrieben. Um zu entscheiden, welcher Sachverhalt konkret bewertet werden soll, stellt die Condition (deutsch: Bedingung) den Kern der Qualitätsanforderung dar. Darin wird festgelegt, welche Umstände erfüllt sein müssen, damit die Anforderung als erfüllt gilt. Die Bedingung, die im Rahmen der Anforderungsauswertung geprüft werden soll, setzt das Strukturmuster *Composite* um. Damit sind sowohl sehr einfache Bedingungen, die nur einen diskreten Wertevergleich oder einen Fuzzy-Term enthalten, als auch komplexe Bedingungen möglich. Die Auswirkung einzelner Bedingungen kann über das Gewicht (englisch: *weight*) und die Art der Komposition (Junktor, englisch: *logical connective*) gesteuert werden. Details zu den Ausprägungen atomarer Bedingungen folgen weiter unten. Ist die Auswertung erfolgt und es stellt sich heraus, dass die Anforderung nicht wie gewünscht erfüllt werden kann, wird über Aktionen (englisch: *actions*) angegeben, mit welchen Maßnahmen darauf reagiert werden soll. Ihre Ausführung erfolgt abhängig vom Ausgang der Auswertung. Da die Ausprägung solcher Adaptionsaktionen sehr vielfältig sein kann, werden sie separat in Abschnitt 6.2 beschrieben. Dabei ist auch die parallele oder zeitlich nacheinander liegende Ausführung voneinander abhängiger Aktionen (Verkettung) möglich. Obwohl das Entwurfsmuster *Composite* Verzweigungen in beliebiger Breite zulässt, wird hier konkret mit *binären Bäumen*

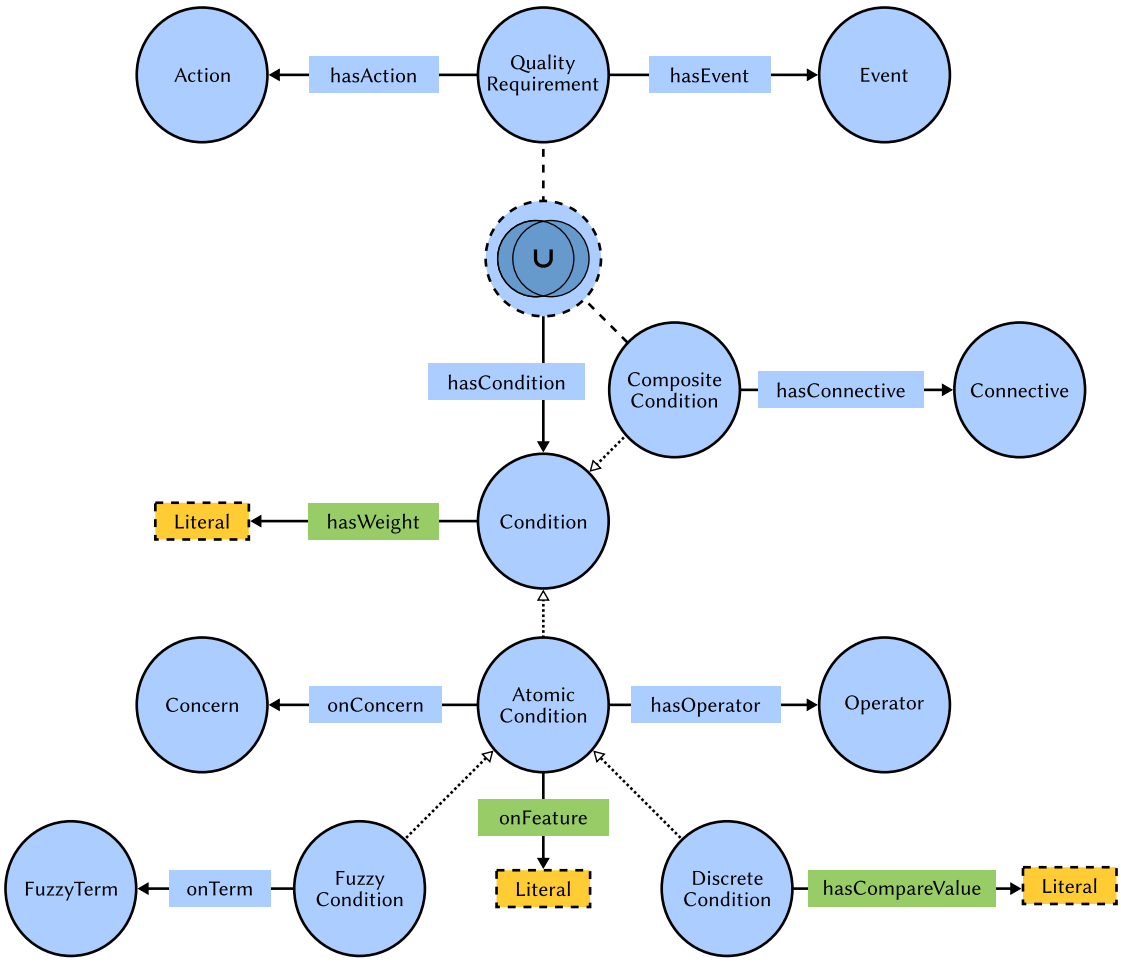

Abbildung 5.3: Metamodell für Qualitätsanforderungen in kompositen Web-Mashups

komponiert. Durch Kombination dieser Verknüpfungen kann eine ebenso große Mächtigkeit erreicht werden wie mit n-Verzweigungen. Als Junktoren stehen *AND* und *OR* zur Verfügung. Sie bestimmen, wie einzelne Auswertungsergebnisse der Bedingungen schrittweise zu einem skalaren Wert verrechnet werden. Für *AND*-verknüpfte Auswertungsergebnisse wird die Minimumsfunktion, für *OR*-Verknüpfungen das Maximum der ermittelten Werte genutzt. Bedingungen können als Fuzzy-Bedingung oder diskrete Bedingung innerhalb einer *Atomic Condition*, dem Blatt-Teil des Composite-Musters, ausgeprägt sein. Fuzzy-Bedingungen drücken unscharfe Teile der Qualitätsanforderung aus und referenzieren über *onTerm* einen zur referenzierten Qualitätseigenschaft passenden *Fuzzy-Term*. Dieser wird in Kombination mit Techniken der Fuzzy-Logik zur automatisierten Auswertung der Qualitätsanforderung genutzt. Diskrete Bedingungen erhalten hingegen exakte quantitative Vergleichswerte (*hasCompareValue*) und dazu passende Operatoren, wie *ist gleich*, *ist kleiner als* oder *beinhaltet*. Verschiedene Strategien im Umgang mit solchen Operatoren werden in Unterabschnitt 5.3.3 diskutiert. Zum Festlegen, welche Wertebelegung einer bestimmten Qualitätseigenschaft zum Vergleich in der Bedingung

genutzt werden soll, wird ein Literal über *onFeature* angegeben. Im Metamodell für Qualitätseigenschaften kommen allerdings keine *Features* vor. Sie sind deren Verallgemeinerung, um auch andere Charakteristika von Kompositionsfragmenten, wie funktionale und technischer Elemente der Kommunikationsschnittstelle, wie *Propertys* und *Capabilitys*, vgl. [RBM16], referenzieren und so in Qualitätsanforderungen verwenden zu können. Somit wird die nahtlose Kombination funktionaler mit Qualitätseigenschaften innerhalb von Qualitätsanforderungen ermöglicht. Features nutzen die Syntax der *SPARQL Property Paths*, siehe [SPARQL, Abschnitt 9: Property Paths], zum Festlegen der Literale. Dabei wird hauptsächlich der Ausdruck / (Schrägstrich für *Sequence Path*) genutzt. Komplex strukturierte Datentypen innerhalb von Features können so innerhalb von Bedingungen direkt referenziert werden. Abhängig vom angegebenen Feature – insbesondere bei Qualitätseigenschaften – können auch *Concerns* spezifiziert werden. Die Qualitätseigenschaft *Zufriedenheit* kann sich beispielsweise auf das *UI* oder die *Funktionalität* von Kompositionsfragmenten beziehen.

Während bei diskreten Bedingungen die Auswertungsergebnisse nur 0 oder 1 annehmen können, liefert die Fuzzy-Auswertung bei der Nutzung entsprechender Fuzzy-Sets feingranulare Basiswerte. Eine Kandidatenkomponente mit einem durch Fuzzy-Auswertung ermittelten Energieverbrauchswert von 0,3 und einer Nutzerzufriedenheit von 0,8 würde eine Gesamtbewertung von 0,8 erhalten, sofern die Bedingungen *OR*-verknüpft sind. In diesem Kontext können diskrete Bedingungen mit Werten von genau 0 oder 1 ohne konzeptionelle Änderungen verrechnet werden. Es sind somit auch komposite Bäume gemischter Bedingungen möglich und sinnvoll auswertbar. Einzelne Bedingungen werden mit einem über *hasWeight* angegebenen Gewicht gegeneinander gewichtet. Die Möglichkeit der Wichtung hat aufgrund ihrer zusätzlichen Komplexität und schwierigen Konfigurierbarkeit, vor allem im direkten Kontakt mit Nutzern, Nachteile, die sie hauptsächlich auf Expertenbedingungen einschränkt. Eine TURTLE-Darstellung des beschriebenen Metamodells für Qualitätsanforderungen bietet Codeausschnitt A.3.

5.3.2 Komposition und Wichtung von Bedingungen

Im Metamodell für Qualitätsanforderungen wurde bereits die Generalisierung auf Features vorgenommen, deren Ausprägung im Kontext dieser Arbeit vor allem Qualitätseigenschaften sind. Andere Ausprägungen, die als Feature innerhalb von Bedingungen auftreten können, sind beispielsweise Wertebelegungen von Propertys als Bestandteil der Komponentenschnittstelle. Damit werden auch Plausibilitätstests ermöglicht, wie etwa das Sicherstellen, dass die Temperatur einer in die Anwendung integrierten Komponente zur *Wettervorhersage* als Temperatur immer weniger als 60 Grad Celsius anzeigt.

Im einfachsten Fall wird in einer diskreten Bedingung ein Vergleich der ermittelten Wertebelegung eines Features mit dem in der Bedingung zugeordneten Vergleichswert vorgenommen. Der Ausgang des Vergleiches entscheidet über das Einleiten einer ggf. zugeordneten Aktion. Diese Art der Bedingung eignet sich für nutzergetriebene Szenarien nur bei hinreichender Kenntnis oder Trivialität des Definitionsbereiches – etwa bei Fünf-Sterne-Bewertungen. Der Bereich für gültige Werte erstreckt sich hierbei von einem Stern bis hin zu fünf Sternen. Eine dazu passende Forderung ist, dass die Nutzerzufriedenheit größer als drei Sterne sein soll. Für mindestens einseitig offene Skalen, beispielsweise beim Energieverbrauch, ist diese Art der Bedingung für menschliche Autoren eher unhandlich. Sie eignet sich jedoch darüber hinaus zum Generieren von Anforderungen aus anderen Anforderungsmodellen oder für das automatisierte

Aushandeln zwischen Systemen. Das Zuordnen von Fuzzy-Termen hat das Ziel, den Nutzerkreis für den Umgang mit Qualitätsanforderungen durch das Angeben von umgangssprachlichen Begriffen in Verbindung mit Features zu vergrößern. Weiß man etwa mit typischen Wertebereichen auf der Watt-Skala für den durchschnittlichen Energieverbrauch nichts anzufangen, hilft die vorab getroffene Zuordnung zu den Termen weiter. Es genügt dann, Paarungen von Termen und Features anzugeben. Sinnvolle Terme für jede Qualitätseigenschaft listet das Referenzmodell in Abschnitt 4.6.

Aus Sicht der Auswertung für Fuzzy-Conditions sind die binären Ergebnisse der diskreten Conditions nur ein Spezialfall. Sie sind somit *kompatibel bezüglich der Komposition*. Die Strategie der Verrechnung der Auswertungsergebnisse wird in Abschnitt 5.5 ausführlich betrachtet. Aus Sicht des Anforderungsautors ergibt sich dadurch der Komfort, diskrete und Fuzzy-Conditions sowie Qualitätseigenschaften und andere Ausprägungen von Features beliebig in einer Anforderung kombinieren und somit eine große Mächtigkeit in Verbindung mit zugeordneten Adaptionsaktionen aufbauen zu können. Vor allem im Empfehlungsprozess kann diese Komplexität iterativ aufgebaut werden, bis das gewünschte Ergebnis gefunden wird.

Zur differenzierten Abwägung einzelner Beiträge der Bewertungen innerhalb kompositer Bedingungen kann ein Gewicht über *hasWeight* als zusätzlicher Faktor zugeordnet werden. Der Vorgabewert ist 1, sodass das ermittelte Auswertungsresultat nicht verändert wird. Da die Auswirkungen kompositer Bedingungen bereits bei nur geringer Komplexität abhängig von den genutzten Junktoren unübersichtlich werden, kommen Gewichte nur bei nicht nutzergetriebenen Anforderungen zum Einsatz. Die Faktoren multiplizieren sich bei benachbarten Bedingungen im binären Baum zu 1, um die Normalisierung des Gesamtergebnisses zu erhalten. Beispielpaare sind {1|1} (Vorgabegewichte), {0,8|1,25} und {0,5|2}. Im letzten Paar wird das erste Auswertungsergebnis, das ein numerischer Erfüllungsgrad zwischen 0 und 1 ist mit 0,5 und das Ergebnis der zweiten Bedingung mit 2 multipliziert. Die Ergebnisse werden dann je nach der Semantik des Junktors zusammengefasst, beispielsweise mit der Minimumsfunktion.

5.3.3 Auswahl geeigneter Operatoren

Zum Festlegen benutzerdefinierter Anforderungen müssen auch die zur Verfügung stehenden Vergleichsoperatoren bei atomaren Bedingungen eine ausreichende Mächtigkeit besitzen. Soll zum Beispiel ein bestimmter Umkreis einer geographischen Position gefordert werden, die Eigenschaftsmodellierung bietet jedoch Längen- und Breitengrade an, ergibt sich ein Problem in der Struktur des Vergleichs. Mit welcher Strategie kann hier die Modellierung der Bedingung erreicht werden bzw. ist dies überhaupt in hinreichender Genauigkeit oder Näherung möglich? Das passende Konstrukt wird durch die Auswahl von Vergleichsoperatoren bereitgestellt. Grundsätzlich existieren folgende Strategien für Operatoren:

Operator als Argument bzw. Plugin: Der Vergleichsoperator wird als Argument der Bedingung durch den Anforderungsautor angegeben. Vorteil hierbei ist, dass sich eine beliebige Komplexität bzw. Genauigkeit erreichen lässt. Nachteil ist, dass eine Sprache definiert bzw. verwendet werden muss, die die gewünschte, im Vorfeld zunächst unbekannte, Ausdrucksstärke erreicht. Im vorangegangenen Beispiel ist dies die Berechnung eines Distanzmaßes für den gewünschten Umkreis bezüglich einzelner geographischer Koordinaten. Zu jedem Operator muss im Evaluator, der die Auswertung der Anforderung

vornimmt und damit den Vergleich ausführt, eine passende Implementierung vorliegen. Vorgefertigte Drittanbieter-Operatoren könnten in einem Repository bereitgestellt und dort ggf. zur Vermarktung angeboten werden.

Viele spezielle Operatoren: Neben einer kleinen Anzahl generischer Vergleichsoperatoren werden auch einige spezielle Exemplare bereitgestellt. Für diesen Mittelweg ergibt sich initial erhöhter Aufwand durch das nicht automatisierbare Erstellen. Für einige weitere Anwendungsfälle, die gehäuft auftreten, stehen dann allerdings Operatoren bereit, die domänenspezifisch einsetzbar sind.

Komposition aus generischen Operatoren: Für die Bedingungen innerhalb von Anforderungen können ausschließlich generische Operatoren genutzt werden. Durch Komposition mehrerer primitiver Vergleiche – und der Verknüpfung mehrerer Bedingungen auf beispielsweise Länge und Breite bei Geokoordinaten – kann zumindest eine Näherung an eine ursprüngliche Anforderung erreicht werden, die einen bestimmten Umkreis einer Geoposition definiert. Ein anderes Beispiel sind Wertebereiche mit der Anforderung einer unteren und oberen Begrenzung. Als Vorteil bei der ausschließlichen Nutzung generischer Operatoren müssen nur diese im Vorfeld definiert werden. Dadurch ergibt sich eine einfache Auswahl und ein geringer initialer Entwicklungsaufwand. Nachteil hingegen ist, dass die Mächtigkeit der originären Anforderung nicht in jedem Fall abbildbar und somit eine Näherung erforderlich ist. Zudem ist die Komposition aufwendig und die kompositen Anforderungen werden unübersichtlich. Dieser Nachteil kann für ausgewählte Fälle durch eine intelligente Werkzeugunterstützung für die Komposition von Bedingungen abgemildert werden.

Selbstverständlich sind diese Strategien kombinierbar. So kann im einfachen Fall auf generische Vergleichsoperatoren zurückgegriffen werden. Wird ein unüblicher Vergleich gewünscht, so kann statt der aufwendigen Komposition über mehrere Teilbedingungen die Variante mit Plugins gewählt werden, sofern ein passendes angeboten wird. Stellt sich heraus, dass eine bestimmte Art des Vergleiches häufiger eingesetzt wird, kann der Betreiber der Mashup-Plattform einen speziellen Operator hinzufügen.

Der Vergleichswert ist laut Metamodell ein singuläres Literal. Eine Erweiterung auf explizit mehrgliedrige Vergleichswerte wird notwendig, wenn Operatoren wie *liegt zwischen* mehrere Argumente fordern. Möglicherweise ist auch die Reihenfolge der Argumente entscheidend. Wird mit einem singulären Vergleichswert gearbeitet, muss eine Strukturierung innerhalb der Zeichenkette erfolgen, etwa über JavaScript Object Notation (JSON). Eine weitere Komfortsteigerung lässt sich durch das Einführen eines *NOT*-Operators zum Modifizieren des Vergleichsoperators bzw. der Vergleichswerte erreichen. Je nach Invertierbarkeit der angebotenen Operatoren führt dies zu einer möglichen Einsparung, verbunden mit einer ggf. schlechteren Lesbarkeit der Bedingung. Werden beispielsweise $<$, $\leq$ und $=$ angeboten, sind durch Negation $\geq$, $>$ und $\neq$ ebenfalls abbildbar.

Explizite Vergleichsoperatoren werden hauptsächlich bei der Nutzung von diskreten Bedingungen benötigt. Die Fuzzy-Auswertung selbst liefert bereits einen Zugehörigkeitsgrad, der mit den diskreten Ergebnissen kompatibel ist, wodurch sie als Standardoperator für Fuzzy-Conditions betrachtet werden kann. Eine Konfiguration ist jedoch auch hier angebracht, wenn etwa mit *mindestens* und *höchstens* in Verbindung mit Termen gearbeitet wird. Auf diese Weise

kann die Fuzzy-Auswertung über die Zugehörigkeitsfunktion mehrerer Terme ausgedehnt werden. Allerdings ist hier ggf. das Bilden einer *Ordnungsrelation* über den Termfunktionen einer Eigenschaft notwendig. Details zur Behandlung dieser Modifikatoren beschreibt Abschnitt 5.5.

5.3.4 Ereignisse zum Auslösen der Anforderungsauswertung

Der Zeitpunkt der Auswertung von Qualitätsanforderungen wird je nach Szenario explizit oder implizit als Ereignis angegeben. Tabelle 5.1 bietet einen Überblick auf die Ereignisse, die in einem Referenzmodell vorgestellt werden. Während der Live-Sophistication tritt der unmittelbare Empfehlungsprozess in den Vordergrund. Dort werden bei der iterativen Verbesserung der Suchergebnisse die schrittweise aufgebauten Bedingungen ausgewertet. Das implizit aus der Art dieses Prozesses abgeleitete Event heißt *OnConditionChange*, da die Änderung der Bedingung der aktuellen Anforderung für das Anbieten von Vorschlägen maßgeblich ist. Aus Sicht des Nutzers ist somit keine Auswahl oder Eingabe von Events im UI notwendig. Wird eine Qualitätsanforderung aus anderen Anforderungsmodellen maschinell erzeugt, so ist auch die unmittelbare Ausführung nach ihrem Bekanntwerden (*Immediately*) sinnvoll.

Soll durch Qualitätsanforderungen die Überwachung von Werten für Qualitätseigenschaften bzw. Features zur Laufzeit einer Mashup-Anwendung erreicht werden, ist das explizite Hinzufügen von Ereignissen zum Konkretisieren des Zeitpunktes für die Anforderungsauswertung notwendig. Im Anforderungsassistenten wird dafür eine Oberfläche angeboten, die neben der Auswahl von Events auch entsprechende Argumente entgegennimmt, vgl. Abschnitt 5.4. Ebenfalls in Tabelle 5.1 werden Beispiele für solche Events mit ihren Attributen aufgeführt. Soll sichergestellt werden, dass eine Bedingung für jede Anwendungskonfiguration gilt, so löst *OnApplicationModelChange* die Auswertung bei jeder Änderung im Kompositionsmodell und seinen Teilmodellen aus. Dies kann ein Hinzufügen oder Entfernen von im Mashup enthaltenen Komponenten oder auch eine Änderung im Screenflow-Modell sein. Hierbei ist zunächst noch keine Verfeinerung mit Argumenten vorgesehen. Dies kann jedoch ergänzt werden. Soll hingegen die Auswertung für Kommunikationsereignisse innerhalb der Anwendung erfolgen, so ist das Event *OnCommunicationEvent* zu wählen. Als Argument werden hier alle im aktuellen Anwendungszustand verfügbaren Events angeboten. Diese können für eine erhöhte Übersichtlichkeit nach Komponenten bzw. Plattform-Events gruppiert werden. Eher domänenspezifisch, jedoch dem mobilen Charakter der Mashup-Plattform Rechnung tragend, wird mit *NearGeoLocation* eine Komfortfunktion als Event angeboten, das die Geokoordinaten und den Umkreis als Argumente zulässt. Damit lassen sich Qualitätsdienstleistungen ortsabhängig realisieren, indem das Erfordernis der Nähe mit anderen nutzerspezifischen Bedingungen kombiniert wird. Ist ein bestimmter Zeitpunkt im Vorfeld bekannt, kann *AtPointInTime* genutzt werden, um beispielsweise bestimmte Tageszeiten in einer automatisierten Qualitätsanforderung abzuprüfen. Als Argument dient hier eine Zeitpunktangabe in der Zukunft. Schließlich lässt sich der Automatisierungsgrad weiter mit *Interval* als Ereignis und der Länge des Intervalls als Argument erhöhen. Hierbei hat weder die Anwendungskonfiguration noch die anwendungsinterne Kommunikation Einfluss auf den Zeitpunkt der Auswertung. Eine TURTLE-Darstellung des Referenzmodells für Events wird in Codeausschnitt B.1 gezeigt.

Tabelle 5.1: Referenzmodell für Ereignisse zum Auslösen der Auswertung von Bedingungen in Qualitätsanforderungen

Name des Events	Beschreibung des Auswertungsumstandes
OnConditionChange	implizit, während der Empfehlung bei Änderung der Anforderungsbedingung
Immediately	implizit, unverzüglich nach Bekanntwerden bzw. Import der Anforderung
OnApplicationModelChange	Änderung in der Komposition, bspw. Entfernen von Komponenten
OnCommunicationEvent	Argument: Auswahl aus *Kommunikationsevents* der aktuellen Anwendung
NearGeoLocation	Argumente: *Geokoordinaten* und *Umkreis*, bspw. ermittelt durch Map-Marker
AtPointInTime	Argument: *Zeitpunkt* in der Zukunft
Interval	wiederkehrende Auswertung, Argument: *Länge des Intervalls*, Auswahl mit Zeit-Slider

5.3.5 Serialisierung von Anforderungsinstanzen

Qualitätsanforderungen als Instanzen des Metamodells in allen Sprachen für RDF serialisiert werden. Von Frameworks wie *Jena* werden vor allem RDF/XML, RDF/JSON, N-Triples, TURTLE sowie JSON-LD unterstützt. Den TURTLE-Beispielcode einer Qualitätsanforderung zeigt Codeausschnitt 5.1. Sie kann durch die Mashup-Infrastruktur ohne zusätzliche Konversion automatisiert verarbeitet werden. Eckige Klammern zeigen an, dass mit anonymen Ressourcen (englisch: *blank nodes*) gearbeitet wird. Die Anforderung enthält eine diskrete Bedingung mit Vergleichswert und Operator ohne Events und Actions. Die Anforderung kann also mit implizit vorbelegten Events genutzt werden, beispielsweise innerhalb des Empfehlungsprozesses.

Codeausschnitt 5.1: Diskrete TURTLE-Qualitätsanforderung, gegen die Kandidaten von Kompositionsfragmenten geprüft werden können

```
1 @prefix : <http://mmt.inf.tu-dresden.de/models/quality/requirement#>.
2 [ a :QualityRequirement;
3   :hasCondition [
4     a :DiscreteCondition;
5     :hasCompareValue "50";
6     :hasOperator :LessThan;
7     :onFeature "Version"
8   ]
9 ].
```

Das nächste Beispiel in Codeausschnitt 5.2 zeigt eine einfache Anforderung unter Verwendung mehrerer Fuzzy-Terme. Hierbei ist zu beachten, dass die Terme im Eigenschaftsmodell zu finden sind, das in Zeile 2 mit dem Präfix qp importiert wird. Das leere Präfix für das Anforderungsmodell aus Zeile 1 bleibt analog zum vorhergehenden Beispiel bestehen.

Codeausschnitt 5.2: Qualitätsanforderung mit mehreren Fuzzy-Termen

```
1 @prefix : <http://mmt.inf.tu-dresden.de/models/quality/requirement#>.
2 @prefix qp: <http://mmt.inf.tu-dresden.de/models/quality/property#>.
3 [ a :QualityRequirement;
4   :hasCondition [
5     a :FuzzyCondition;
6     :onFeature "Version";
7     :onTerm qp:High, qp:Medium
8   ]
9 ].
```

Komplexe Bedingungen machen sich das Entwurfsmuster Composite zunutze. Diskrete und Fuzzy-Bedingungen können in dem dabei entstehenden binären Baum gemeinsam auftreten. Codeausschnitt 5.3 zeigt eine solche Qualitätsanforderung mit kompositer Bedingung und gemischten Typen von Bedingungen. Die beiden Teilbedingungen sind hierbei durch den Junktor *AND* verbunden. Dieser macht die diskrete Bedingung *obligatorisch*, da durch die Aggregation mittels des *Minimums* der Erfüllungsgrad der Gesamtbedingung auf 0 sinkt, sollte die diskrete Bedingung nicht erfüllt sein.

Codeausschnitt 5.3: Qualitätsanforderung mit kompositer Bedingung

```
@prefix : <http://mmt.inf.tu-dresden.de/models/quality/requirement#>.
@prefix qp: <http://mmt.inf.tu-dresden.de/models/quality/property#>.
[ a :QualityRequirement;
   :hasCondition [
      a :CompositeCondition;
      :hasCondition [
         a :FuzzyCondition;
         :onFeature "Version";
         :onTerm qp:Medium
      ];
      :hasCondition [
         a :DiscreteCondition;
         :hasCompareValue "50";
         :hasOperator :LessThan;
         :onFeature "Version"
      ];
      :hasConnective :And
   ]
].
```

In Codeausschnitt 5.4 wird zusätzlich ein Concern verwendet. Darüber hinaus sind hier Event und Action explizit angegeben, wie es etwa bei Anforderungen zur Überwachung von Qualitätseigenschaften zur Laufzeit einer Mashup-Anwendung üblich ist. Es handelt sich hierbei um ein Intervall mit dem zusätzlichen Argument der zeitlichen Länge. Die Action stammt aus dem entsprechenden Referenzmodell – vgl. Unterabschnitt 6.2.2 – und trägt als Argument eine Nachricht, die im Falle der Verletzung der Bedingung als Popup angezeigt wird.

Codeausschnitt 5.4: Qualitätsanforderung mit Concern, Event und Action

```
@prefix : <http://mmt.inf.tu-dresden.de/models/quality/requirement#>.
@prefix qp: <http://mmt.inf.tu-dresden.de/models/quality/property#>.
[ a :QualityRequirement;
   :hasCondition [
      a :DiscreteCondition;
      :hasCompareValue "300";
      :hasOperator :GreaterThan;
      :onFeature "ResponseTime";
      :onConcern qp:Service
   ];
   :hasEvent [
       a :Interval;
       :hasInterval "12000"
   ];
   :hasAction [
       a :ShowPopup;
       :hasMessage "You better check your network connection."
   ]
].
```

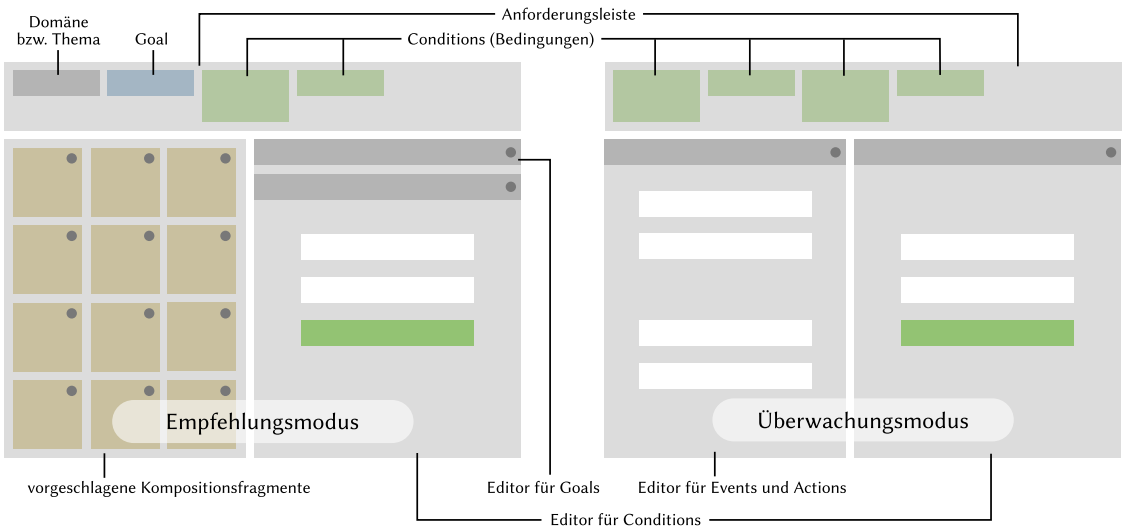

Abbildung 5.4: Schematische Darstellung der Modi des Anforderungsassistenten

5.4 Erzeugen und Bearbeiten von Qualitätsanforderungen

Hauptziel des Spezifizierens von Qualitätsanforderungen während der Live-Sophistication von Web-Mashups ist die möglichst passgenaue Übertragung der Intention des Nutzers in Instanzen des vorgestellten Anforderungsmetamodells. Daher sind die angebotenen Interaktionstechniken und visuellen Metaphern essenziell für den Erfolg des Prozesses der qualitätsbewussten Mashup-Nutzung. Die im Überwachungsszenario benötigten Nutzerschnittstellen müssen zusätzliche Anforderungen erfüllen, die durch Expertenfähigkeiten im Formulieren von Anforderungen mit Events und Actions entstehen. Dieser Abschnitt beschreibt, wie das nutzergetriebene *Erzeugen* und *Bearbeiten* von Qualitätsanforderungen in der Laufzeitumgebung für komposite Web-Mashups erfolgt. Dafür bietet ein *Anforderungsassistent* (englisch: *requirements wizard*) mehrere Modi an, um den Fähigkeiten seiner Nutzer entsprechend Modellinstanzen für Qualitätsanforderungen zu verwalten. Hierbei wird auch die Anbindung an das Eigenschaftsmodell und das Wiederverwenden favorisierter Anforderungen aus dem Nutzerkontext betrachtet. Abbildung 5.4 zeigt eine schematische Darstellung der beiden Modi des Anforderungsassistenten für *Empfehlung* und *Überwachung*. Dort fällt zunächst auf, dass die Anforderungsleiste mit der Darstellung aktiver Conditions sowie der Editor für diese Conditions gemeinsame Elemente in beiden Modi sind. Domäne und Goal repräsentieren dabei Anforderungen, die nur im Empfehlungsmodus benötigt werden. Der Empfehlungsmodus zeigt exklusiv die vorgeschlagenen Kompositionsfragmente, die im Überwachungsmodus nicht anwendbar sind. Umgekehrt bietet der Überwachungsmodus einen Editor für Actions und Events, die während der Empfehlung implizit gesetzt sind. Außerdem entfallen bestimmte, für die Empfehlung nützliche Anforderungstypen. Die Nutzerschnittstellen zum Festlegen und Anpassen aller Elemente der Qualitätsanforderungen werden nun detailliert vorgestellt.

5.4.1 Empfehlungsmodus

Eine der Hauptentwicklungsaufgaben bei der Live-Sophistication in Szenario ① ist das Finden und Integrieren neuer Komponenten zur gewünschten Anwendung. Anfangs sollen damit neue Anwendungsfunktionalitäten aufgebaut, später deren Umfang ergänzt werden. Um zur Anforderung passende Komponenten, fertige Anwendungen oder Anwendungsfragmente zu erhalten, wird ein Empfehlungssystem eingesetzt, das sowohl die über einen Assistenten spezifizierten Qualitätsanforderungen als auch funktionale Ziele (englisch: *goals*) berücksichtigt, vgl. [Rad+12]. Wird nicht mit einer vorhandenen Anwendung, sondern von Neuem gestartet, so kann als übergeordnetes Goal eine Domäne bzw. ein Thema – wie beispielsweise *Finanzen* oder *Tourismus* – ausgewählt werden. Die Vorschriften, die die Bewertungen und damit die Sichtbarkeit und Reihenfolge der empfohlenen Kompositionsfragmente bestimmen, werden in Abschnitt 5.5 erklärt.

Exklusiv werden im Empfehlungsmodus vorgeschlagene Kompositionsfragmente angezeigt, die absteigend nach ihrem Auswertungsergebnis sortiert sind. Neben einem Icon des Kompositionsfragments werden noch wichtige Metadaten im Überblick visualisiert. Da die Auswertung der über die Conditions festgelegten Qualitätsanforderungen nach jeder Änderung erneut erfolgt, steht immer eine aktuelle Ansicht der empfohlenen Kompositionsfragmente zur Verfügung. Dies erleichtert die iterative Anpassung der Anforderung, um über den zeitlich unmittelbaren Vergleich zur passendsten Empfehlung zu kommen. Wird ein Kompositionsfragment vom Nutzer ausgewählt, so verlässt er den Empfehlungsmodus, damit auch den Anforderungsassistenten und geht zunächst in den Nutzungsmodus über. Dort beginnt er mit dem Laden der ausgewählten Komponente bzw. Anwendung eine neue Aufgabe zu bearbeiten. Alternativ verbessert er die Werkzeugunterstützung für eine bereits begonnene Aufgabe durch das Hinzufügen einer weiteren Mashup-Komponente. Das explizite Festlegen von Events und Actions ist im Empfehlungsmodus nicht erforderlich, da diese beiden Angaben bereits durch den Entwicklungsprozess vorgegeben sind, vgl. dazu auch Tabelle 5.1.

5.4.2 Überwachungsmodus

In Szenario ② steht die Überwachung von Qualitätsanforderungen im Fokus. Der für diese Aufgabe geschaffene Überwachungsmodus des Anforderungsassistenten bietet verschiedene *Scopes*, d. h. Geltungsbereiche, an. Zum einen kann der Assistent im Kontext einer laufenden Mashup-Komponente aktiviert werden, zum anderen steht er im Kontext der kompositen Anwendung für alle Komponenten zur Verfügung. Die Scope-Awareness wird durch ein entsprechendes Label im Assistenten gewährleistet. Für jeden Scope sind dann jeweils die geltenden Anforderungen in der Anforderungsleiste sichtbar.

Exklusiv im Überwachungsmodus steht jeweils ein Editor für Events und Actions zur Verfügung. Sie werden den Blöcken der Conditions in der Anforderungsleiste zugeordnet. Im Editor für Events stehen im Wesentlichen die möglichen Einträge aus dem Referenzmodell in Tabelle 5.1 mit ihren Argumenten zur Auswahl. Dafür sind zum Event passende Eingabeformen – wie ein Zeit-Slider für ein Intervall – verfügbar, vgl. Unterabschnitt 7.5.1. Analog dazu erfolgt die Auswahl der Aktionen nach den Vorgaben des Modells der Aktionen in Abschnitt 6.2. Auswahlmöglichkeiten für Argumente werden ggf. aus der aktuellen Anwendungskonfiguration generiert, wie etwa eine Liste aller Kommunikationsevents, die von den integrierten Mashup-

Komponenten angeboten werden. Soll beispielsweise für eine POI-Komponente die Antwortzeit des angebundenen Web-Services nicht einmalig, sondern wiederholt überwacht werden, wird zunächst im Kontext dieser Komponente der Anforderungsassistent im Überwachungsmodus geöffnet. Dort erfolgt dann das Festlegen der Condition mit der Qualitätseigenschaft *Response Time* und einem entsprechenden Vergleichswert und Operator. Über die Condition, die in der Anforderungsleiste als Block auftaucht, kann nun das Intervall als Event ausgewählt und über einen Zeit-Slider konfiguriert werden. Die zuzuordnende Adaptionsaktion – beispielsweise der Austausch der Komponente gegen eine Alternative aus dem Repository – wird direkt im Anschluss über den Editor für Actions ausgewählt und konfiguriert. Geltende Anforderungen im Überwachungszustand werden über ein Symbol am Condition-Block vermerkt. Sie können dort auch wieder aus der Anforderung entfernt werden.

5.4.3 Modifikationspunkte kompositer Bedingungen

In beiden Modi steht ein Editor für Conditions zur Verfügung, der die Komposition von binären Bäumen für Bedingungen mit den im Metamodell vorgesehenen Elementen erlaubt. In Abbildung 5.5 wird die iterative Komposition von Conditions am Beispiel aller möglichen Erweiterungspunkte strukturell dargestellt. Nach dem initialen Erstellen der *Condition 1* gibt es nur eine Möglichkeit der Erweiterung. Die Junktoren in der Abbildung sind dabei frei gewählt, in jedem Fall aber kommutativ. Die Reihenfolge der Zuordnung von Conditions zu einem Junktor – hier dargestellt als *oben* und *unten* – spielt also keine Rolle. Wurde die *Condition 2* über den Junktor *AND* hinzugefügt, ergibt sich eine komposite Condition, die drei Erweiterungspunkte besitzt. An diesen Punkten soll nun eine dritte Condition mit *OR* ergänzt werden. Der obere Fall ergänzt an Condition 1, der mittlere Fall an Condition 2 und der untere Fall am Junktor *AND*. Alle drei daraus resultierenden Condition-Blöcke bieten selbst jeweils fünf Erweiterungspunkte. Beim iterativen Hinzufügen von Conditions in diesen binären Baum sind also immer $2n - 1$ Erweiterungspunkte im n-ten Erweiterungsschritt verfügbar. Neben dem Zusammensetzen komplexer Bedingungen ist auch das Löschen einzelner Conditions möglich. Ohne die Struktur des Bedingungsbaumes zu verändern, lassen sich die Junktoren zwischen den Optionen *AND* und *OR* umschalten. Bestätigte Condition-Blöcke erscheinen in der Anforderungsleiste. Die Eingabefunktionen des Editors für Condition, der in beiden Modi Anwendung findet, werden in Abschnitt 7.5 vorgestellt.

5.4.4 Favorisierte Anforderungen und Qualitätsprofile

Wiederkehrend genutzte Qualitätsanforderungen werden in einem Kontextdienst gespeichert, sodass sie dem angemeldeten Nutzer wieder angeboten werden können. Um eine Anforderung als Favorit zu markieren, wählt der Nutzer ein Favoritensymbol, das in einem Hover-Menü des Anforderungsblocks in der Anforderungsleiste des Assistenten erscheint. Es besitzt eine Toggle-Fähigkeit, sodass Favoriten einfach an- und abgewählt werden können. Im Kontextmodell werden favorisierte Anforderungen dem eindeutigen Nutzerbezeichner zugeordnet, der über die Laufzeitumgebung bereitgestellt wird. Werden dem Nutzer zuvor favorisierte Anforderungen angeboten, kann er auch entscheiden, dass diese für die aktuelle Sitzung bzw. Kompositionsaufgabe nicht berücksichtigt werden sollen. Das Speichern und Abrufen favorisierter Anforderungen über die verteilte Laufzeitumgebung und den Kontextdienst ist in Abbildung 5.6 dargestellt.

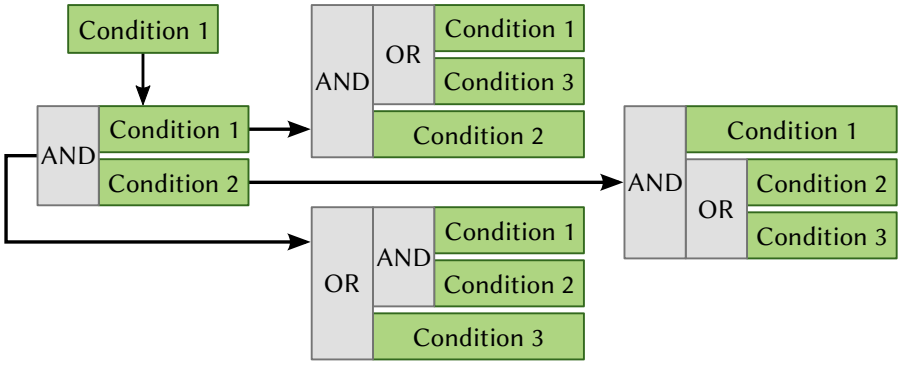

Abbildung 5.5: Iterative Komposition von Conditions über Erweiterungspunkte

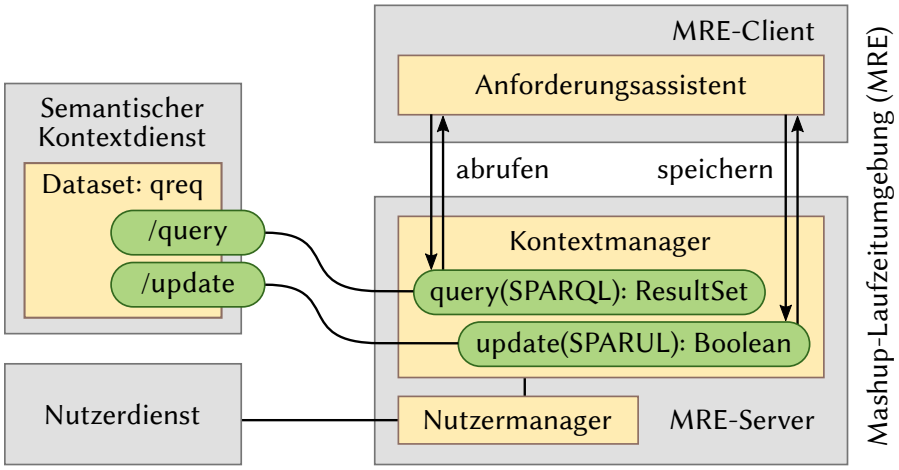

Abbildung 5.6: Verwaltung und Nutzung favorisierter Qualitätsanforderungen in der verteilten Laufzeitplattform für Mashups unter Nutzung eines Kontextdienstes

Der Nutzer des Anforderungsassistenten erhält beim Anmelden an der Plattform über den Nutzerdienst eine Nutzerkennung, die auch im Kontextmodell zum Zuordnen seiner gespeicherten Anforderungen genutzt wird. Beim initialen Speichern einer Anforderung wird diese vom Anforderungsassistenten an den Kontextmanager der Laufzeitumgebung übermittelt. Zur Übertragung innerhalb der MRE werden etablierte Systemkommandos genutzt. Der Kontextmanager speichert die Anforderung über die Schnittstelle des Kontextdienstes in das Kontextmodell des angemeldeten Nutzers. Soll eine gespeicherte Anforderung im Anforderungsassistenten bereitgestellt werden, so ruft der Kontextmanager analog die zur Nutzerkennung gespeicherten Anforderungen ab und leitet diese zur Visualisierung weiter.

Sofern ein gesamter Anforderungsblock mit Bedingungen durch die entsprechende Schaltfläche entfernt wird, erfolgt neben dem Entfernen der Bedingung aus dem aktuellen Kompositionskontext kein Löschen des Favoriten aus dem Kontextmodell. Der Favorit wird in der nächsten Sitzung wieder angeboten. In bestimmten Fällen nutzergetriebener Interaktion werden zuvor favorisierte Anforderungen auch im Kontextmodell aktualisiert:

Abbildung 5.7: Beispiele für Qualitätsprofile als vorgefertigte Anforderungen mit Label-Text und visueller Unterstützung

- Erweitern einer Bedingung mit einer neuen Teilbedingung und deren Bearbeitung
- Entfernen einer bestehenden Teilbedingung
- Verändern des Typs des Junktors (*AND* ↔ *OR*)

Ein Ziel dieser Arbeit ist das Erleichtern des Zugangs zum Spezifizieren von Qualitätsanforderungen für Web-Nutzer ohne Programmiererfahrung. Diese Eignung für die Zielgruppe soll insbesondere im Szenario ① während der Empfehlung von Kompositionsfragmenten erreicht werden. Die soeben vorgestellte Komposition von Bedingungen ermöglicht zwar eine hohe Komplexität, mit dieser sinkt jedoch auch die Übersichtlichkeit. *Qualitätsprofile* sind vordefinierte Anforderungen, die zusätzlich zur Möglichkeit des grundsätzlichen Aufbaus von Bedingungen und der Zuordnung von Events und Actions beliebte Erfordernisse zusammenfassend unter einem Label anbieten. In Abbildung 5.7 werden einige Beispielprofile gezeigt. Wer *save energy* wählt, kann davon ausgehen, dass ein Bedingungsbaum in der zugehörigen Anforderung hinterlegt ist, der zum Profil passende Qualitätseigenschaften referenziert, die von Anforderungsexperten ausgewählt wurden. *premium rating* lässt nur hohe und sehr hohe Nutzerbewertungen der Kompositionsfragmente zu. Einträge, die dem Repository erst in den letzten Wochen hinzugefügt wurden, werden unter *recently added* empfohlen. Bei *fast reaction* wird der Fokus auf die Antwortzeit angebundener Web-Services gesetzt. Schließlich bietet *trustworthy* nur Inhalte einer Liste zertifizierter Autoren.

Aus struktureller Sicht sind Qualitätsprofile gespeicherte Qualitätsanforderungen mit zusätzlichen Metadaten, wie einem Label und einem illustrativen Bild. Für Überwachungsszenarien können solche Profile selbstverständlich die gesamte Mächtigkeit des Anforderungsmetamodells nutzen und somit auch Events und Actions enthalten. Die Profile sollen vornehmlich von Menschen genutzt werden, denen es zu umständlich ist, eigene Anforderungen zu komponieren. In bestimmten Situationen kann es jedoch erforderlich werden, ein ausgewähltes Profil anzupassen. Hierfür wird in die strukturelle Sicht gewechselt, die aus dem Anforderungsassistenten bekannt ist. Dort kann dann der Bedingungsbaum wie üblich bearbeitet und ggf. die Zuordnung von Events und Actions vorgenommen werden. Das Erstellen eines neuen Profils aus einem vorhandenen Profil, das bearbeitet wurde, erfolgt dann durch einen Experten, der auch ein Label und ein Bild hinzufügt.

5.5 Auswertung von Qualitätsanforderungen

Die Auswertung von Qualitätsanforderungen liefert die Grundlage für die Überwachung von Qualitätseigenschaften und anderen Features zur Laufzeit von kompositen Web-Mashups. Sie ist überdies entscheidend für den Empfehlungsprozess von Kompositionsfragmenten während der

nutzergetriebenen Anwendungsentwicklung. Die Automatisierung im Auswertungsprozess benötigt dabei den maschinellen Zugriff auf Vergleichswerte der in den Anforderungen genutzten Features. Bei diskreten Bedingungen werden erwartete Werte mit tatsächlichen Werten verglichen. Bei der Fuzzy-Auswertung tritt an die Stelle der Kombination aus Vergleichswert und Operator ein Fuzzy-Term. Dieser Abschnitt beschreibt Rahmenbedingungen für die automatisierte Auswertung von Qualitätsanforderungen, den Auswertungsalgorithmus und die Besonderheiten der Abstimmung verschiedener Auswertungsorte sowie die Schnittstelle zum Auslösen konfigurierter Aktionen.

5.5.1 Rahmenbedingungen für die Auswertung

Dank der vielseitigen Einsetzbarkeit der anpassbaren Qualitätsanforderungen werden diese der Auswertung aus unterschiedlichen Quellen zugeführt. Neben dem Festlegen über den Anforderungsassistenten in den Szenarien ① und ② kommen in Szenario ③ auch existierende Modelle – wie die der Kompositionsfragmente (SMCDL und MCM) und externe Anforderungsdokumente, Unternehmensrichtlinien oder Aufgabenmodelle – als Quellen in Frage. Indirekt können Anforderungen jedoch auch aus Kontextmodellen von Nutzern stammen, die sie als Favoriten in die Plattform bringen. Wichtige Rahmenbedingung ist außerdem die Verfügbarkeit aktueller Vergleichswerte für die referenzierten Qualitätseigenschaften. Hierbei ist zu beachten, dass je nach Typ der Wertermittlung die Häufigkeit der Wertänderung sehr stark variiert. Während sich Metadaten von Modellen aus den Repositorys generell sehr selten ändern, unterliegen Sensordaten, die laufend neu in der Laufzeitumgebung oder indirekt durch die beteiligten Geräte oder angebundenen Dienste bestimmt werden, sehr häufigen Änderungen. Vor allem bei der Kopplung von Qualitätseigenschaften an Sensorwerte kann eine vorherige Kontextaggregation sinnvoll sein. Im Bereich verteilter Mashups beschäftigen sich Mroß u. a. mit dieser Problemstellung [MM14]. Diese aggregierten Eigenschaften können dann als Erweiterung des Referenzmodells für Qualitätseigenschaften genutzt werden.

5.5.2 Verteilter Auswertungsprozess

Wie in Abbildung 5.1 angedeutet, erfolgt die Auswertung von Qualitätsanforderungen an mehreren Stellen in den *Evaluatoren*. Diese sind auch in Abbildung 5.8 wiederzufinden, welche das Vorgehen beim Auswerten von Qualitätsanforderungen mit Bedingungen an gemischten Eigenschaftstypen zeigt. Dafür wird als Eingabe eine Anforderung mit drei verschiedenen Bedingungen genutzt. Die Details der Komposition dieser Bedingungen spielen dabei eine untergeordnete Rolle. Der Wert von *c1* wird durch Messung in der Laufzeitumgebung durch Sensorik ermittelt. *c2* stellt eine bereitgestellte Qualitätseigenschaft dar und *c3* ist ein Ergebnis aggregierter Einzelwerte, die im Repository verwaltet werden.

Im praktischen Einsatz werden die Anforderungskonfigurationen in vielen Fällen nur einen der beiden Evaluatoren in Anspruch nehmen. Während der Empfehlung werden typischerweise nur gesammelte und bereitgestellte Eigenschaften genutzt, weil das als Kandidat zu bewertende Kompositionsfragment zu diesem Zeitpunkt noch nicht in der Laufzeitumgebung ausgeführt wird. Im Überwachungsszenario sind es vor allem die gemessenen Eigenschaften, möglicherweise auch die gesammelten Werte mit langsamerer Änderungsrate und theoretisch auch die bereitgestellten Eigenschaften, die sich jedoch kaum ändern.

Zunächst gelangt die Qualitätsanforderung in den Qualitätsmonitor, der Teil der MRE ist. Im Überwachungsszenario wird nach Kenntnisnahme einer Anforderung zunächst auf das Eintreten eines konfigurierten Events gewartet. Ist der damit verbundene Auswertungszeitpunkt gekommen, erfolgt die Auswertung der Bedingung, die sich im Beispiel aus drei atomaren Bedingungen zusammensetzt. Für jede Bedingung wird der Typ der Wertermittlung der referenzierten Features festgestellt. Sind unter den auszuwertenden Bedingungen solche, deren Wertebelegung von Features durch Messung in der Laufzeitumgebung ermittelt wird, erfolgt in *Schritt 1* zunächst die Übergabe der Bedingung – in der Abbildung ist es *c1* – an den Evaluator in der Laufzeitumgebung. Diese beschafft sich den aktuell gültigen Wert über die jeweilige Sensorik, vgl. Unterabschnitt 4.5.3. Der Evaluator berechnet nun für *c1* das Ergebnis (englisch: *result*) *r1*. Der Wertebereich für alle Ergebnisse erstreckt sich dabei von 0 bis 1. Das erste Ergebnis wird anschließend an den Qualitätsmonitor zurückgeliefert. Sind außerdem keine weiteren messbaren Features in Bedingungen der Qualitätsanforderung enthalten, so wird mit *Schritt 2* fortgefahren. Die restlichen Bedingungen werden zur weiteren Auswertung an das Repository übergeben. Dabei sind die bereits vorliegenden Ergebnisse – in der Abbildung ist es *r1* – in der Anfrage enthalten. Dies ist notwendig, damit Zwischenergebnisse korrekt miteinander verrechnet werden können. Die Art und Weise der Verrechnung ergibt sich aus der Komposition der Bedingungen und der Wahl der Junktoren. Der Evaluator im Repository berechnet nun für alle ausstehenden Bedingungen – hier *c2* und *c3* – die Ergebnisse. Der Vergleichswert für *c2* stammt aus dem Komponentenmodell der Kandidaten. Er wurde dorthin aus den SMCDLs der registrierten Komponenten importiert. Zur Auswertung von *c3* werden die durch die Infrastruktur im Repository gesammelten Werte aggregiert. Sind alle Ergebnisse für die Bedingungen ermittelt, wird im letzten Schritt des Evaluators noch eine Aggregation der Ergebniswerte vorgenommen. Dort wird aus *r1* und *r2* ein Wert *r4* berechnet. Schließlich bildet *r5* das Gesamtergebnis für die Anforderung, indem *r4* und *r3* verrechnet werden. Die Kompositionsstruktur komplexer Bedingungen ist von Anfang an in der Anforderung festgeschrieben. In der Abbildung wird auf die Darstellung der Beziehungen zwischen den Bedingungen zur visuellen Vereinfachung verzichtet. Sind alle Ergebnisse ermittelt bzw. berechnet, wird die gesamte Datenstruktur zurück an den Qualitätsmonitor in der Laufzeitumgebung übermittelt. Dieser führt nun abhängig vom Gesamtergebnis – im Beispiel *r5* – die in der Qualitätsanforderung konfigurierten Aktionen aus. Die Ausführung wird angestoßen, sobald ein vorher festgelegter Schwellwert unterschritten wird, die Anforderung also hinreichend stark *verletzt* ist. Dabei bestimmt die gewählte Aktion, ob die Laufzeitumgebung im Zuge der Ausführung weitere Bestandteile der Infrastruktur benötigt, wie etwa das Repository beim Anstoßen eines Empfehlungsprozesses. Die Kommunikation zwischen der MRE – die Schritt 1 ausführt – und dem Repository – das für Schritt 2 verantwortlich ist – erfolgt über die REST-Schnittstellen, die für das Auswerten im CoRe geschaffen wurden, vgl. Unterabschnitt 7.4.2.

5.5.3 Auswertungsalgorithmus

Die Auswertung folgt in den beiden Evaluatoren grundsätzlich demselben Algorithmus. Dieser benötigt als Eingabe den Bedingungsbaum aus einer Qualitätsanforderung sowie eine Menge an Kompositionsfragmenten mit belegten Werten als Kandidaten. Die Anforderung mit den kompositen Bedingungen liegen als Instanz des Metamodells für Qualitätsanforderungen gemäß Unterabschnitt 5.3.1 vor. Die Kandidaten hält das Repository als Komponenten- bzw. An-

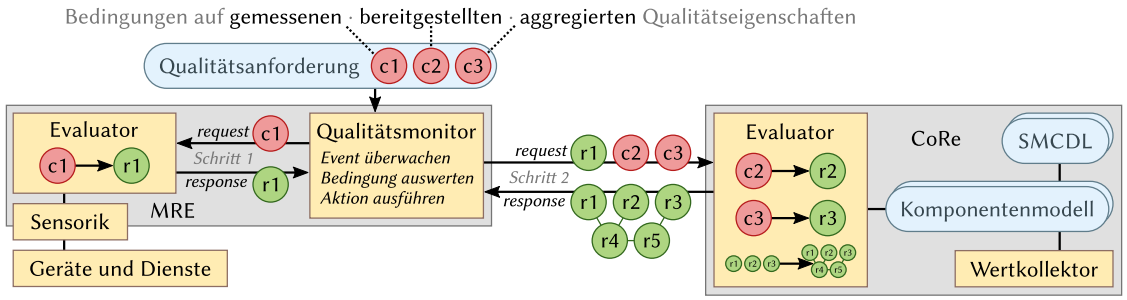

Abbildung 5.8: Auswertungsprozess für Qualitätsanforderungen mit mehreren Bedingungen (*c1* bis *c3*) auf verschiedenen Eigenschaftstypen und den Auswertungsergebnissen *r1* bis *r5*

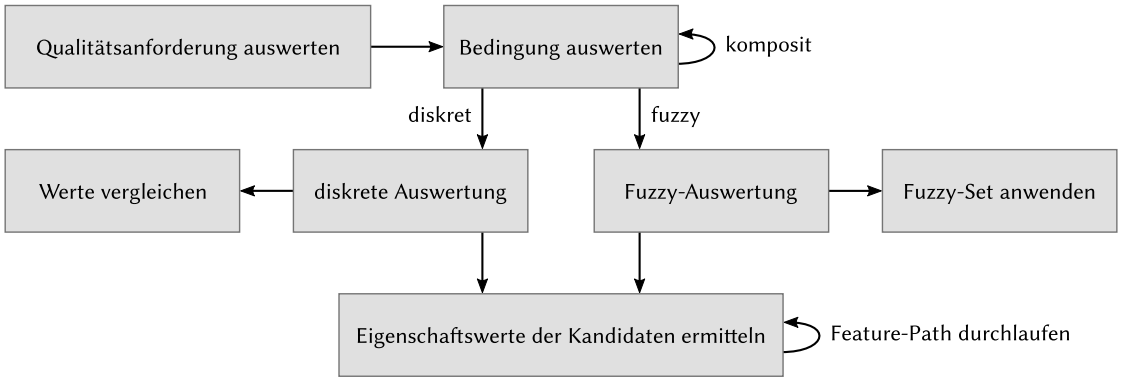

Abbildung 5.9: Auswertungsprozess für Qualitätsanforderungen im Evaluator

wendungsmodelle mit ihren Eigenschaften als Instanzen des Metamodells für Qualitätseigenschaften bereit. In Abbildung 5.9 werden die wichtigsten Schritte des Auswertungsprozesses gezeigt, der für die zu überprüfenden Kandidaten jeweils eine Bewertung liefert. Die in Frage kommenden Kompositionsfragmente – bzw. allgemein: *PropertyCarryingEntitys* – stammen beispielsweise aus dem CoRe. Ihre Propertys stehen während des Auswertungsprozesses als Vergleichspartner zur Verfügung. Zunächst gelangt eine Qualitätsanforderung zum Evaluator, wo die Wurzelbedingung im ersten Schritt extrahiert wird. Sie wird zur rekursiven Verarbeitung an den Schritt *Bedingung auswerten* übergeben. Hier erfolgt die Unterscheidung nach Typen von Bedingungen. Handelt es sich um eine *komposite* Bedingung, wird rekursiv weiter verfahren. Ist die Bedingung *fuzzy* oder *diskret*, kümmern sich die entsprechenden Methoden fallspezifisch um die weitere Behandlung. Bei einer atomaren Bedingung werden allerdings in jedem Fall die *Eigenschaftswerte der Kandidaten ermittelt*. Dort wird der *Feature-Path* traversiert und schließlich der in der Bedingung spezifizierte Property-Wert jedes Kandidaten geliefert. Dabei wird auch der *Concern* berücksichtigt, der neben dem Pfad von Typen charakteristisch für einen bestimmten Eigenschaftswert ist.

Im diskreten Fall wird unter *Werte vergleichen* ein binärer Erfüllungsgrad aufgrund des in der Bedingung geforderten Vergleichswertes, des spezifizierten Operators – beispielsweise *Contains* oder *LessThan* – sowie des tatsächlichen Property-Wertes ermittelt. Im Fuzzy-Fall wird zunächst

versucht, ein passendes *Fuzzy-Set* zu finden, das über eine Zugehörigkeitsfunktion definiert ist. Sofern mehrere *Terme* der Bedingung beigefügt wurden, ist es ein Fuzzy-Set pro Term. Aus den in *Fuzzy-Set anwenden* errechneten Funktionswerten wird nun das *Maximum* – unter Annahme der *OR*-Verknüpfung der gegebenen Terme – gebildet. Dieser maximale *Grade of Membership*, welcher kontinuierlich zwischen 0 und 1 liegen kann, ist gleichzeitig der Erfüllungsgrad der atomaren Anforderung. Innerhalb von *Bedingung auswerten* werden die hierarchisch gesammelten Bewertungen dem angegebenen *Junktor* entsprechend verrechnet, vgl. Unterabschnitt 5.3.1, sodass für die ausgewertete Qualitätsanforderung ein singulärer Wert entsteht. Schrittweise werden so die Einzelergebnisse aggregiert. Die Datenstruktur des Gesamtergebnisses enthält für jeden Kandidaten auch die Teilergebnisse, die für die Awareness im Empfehlungsmodus benötigt werden.

5.6 Zusammenfassung

Der Umgang mit Qualitätsanforderungen in Web-Mashups bildet den konzeptionellen Kern dieser Arbeit. Die zu Beginn dieses Kapitels aufgezeigten Herausforderungen spannen den Bogen von den Nutzererwartungen bis hin zu den infrastrukturellen Rahmenbedingungen der Mashup-Plattform. Dabei werden die Modelle, Werkzeuge und Module – die einerseits zum Festlegen und andererseits zum Auswerten von Qualitätsanforderungen benötigt werden – in die Referenzarchitektur auf Basis der Mashup-Plattform CRUISE eingeordnet. Auf Basis dieser grundlegenden Infrastruktur aus *Anforderungsassistenten* und *Evaluatoren* entstehen weitere Teile der Infrastruktur, die über Erweiterungen der verteilten Laufzeitumgebung die Nutzung *favorisierter Anforderungen und Qualitätsprofile* erlauben. Damit wird ein Forschungsbeitrag für These 5 geliefert, vgl. Unterabschnitt 1.2.3. Konkrete Qualitätsanforderungen, die in der Plattform erzeugt und verarbeitet werden, folgen einem *Metamodell*, das alle wichtigen strukturellen Charakteristika beschreibt und mit dem Modell für Qualitätseigenschaften interagiert. Damit wird ein Vorteil aus der vereinheitlichten Modellierung qualitätsrelevanter Basisgrößen gezogen, vgl. These 1 in Unterabschnitt 1.2.1. Wichtige Teilkonzepte sind der Anforderungsaufbau nach dem ECA-Prinzip, die komposite Struktur von Bedingungen und die kontextabhängige Operatorzuweisung für Vergleiche bei diskreten und Fuzzy-Bedingungen. Zentral für das Festlegen von Qualitätsanforderungen ist der Anforderungsassistent, der in zwei Modi zur *Empfehlung* und *Überwachung* viele Anwendungsfälle des Erzeugens von Anforderungen abdeckt. Neben strukturellen Überlegungen zur Komposition und Bearbeitung der Bedingungsbäume und der Zuordnung von Events und Actions nimmt dieses Kapitel schließlich Stellung zum Auswertungsprozess. Dieser beinhaltet einen verteilten Auswertungsprozess, der in den Evaluatoren den *Auswertungsalgorithmus* beinhaltet. Zum Unterscheiden der vielfältigen Auswertungsgelegenheiten beschreibt dieses Kapitel ein *Referenzmodell für Ereignisse*, das insbesondere zum Ziel der *automatisierten Auswertung* beiträgt, vgl. These 2 in Unterabschnitt 1.2.2. Die möglichen Adaptionsaktionen, die sich einer Anforderungsauswertung anschließen können, sind Gegenstand des nächsten Kapitels. Im Rahmen des Entwicklungsprozesses in Abschnitt 6.1 wird dort das Zuführen der Ergebnisse der Auswertung von Qualitätsanforderungen in die Empfehlung und andere Arten der Nutzung diskutiert.

Die strukturierte Modellierung von Anforderungen erlaubt die leichte Erweiterbarkeit des Konzeptes auf andere Domänen und Situationen durch neue bzw. angepasste Referenzmodel-

le aufgrund der Trennung vom Metamodell. So können beispielsweise neue Eigenschaften und Fuzzy-Terme hinzugefügt werden. Auch die Serialisierung und damit der Austausch und die Validierung sind durch die konsequente Nutzung von RDF als Basisformat auch für die Instanzdaten unproblematisch, da eine breite Unterstützung mit Plattformen und Werkzeugen vorherrscht. Dieses Kapitel schließt somit durch das Ermöglichen der *automatisierten Auswertung anpassbarer Qualitätsanforderungen* eine große konzeptionelle Lücke, die während der Analyse bestehender Forschungsarbeiten identifiziert wurde. Das Anforderungskonzept ist dabei nicht auf das Formulieren von Bedingungen auf Qualitätseigenschaften beschränkt, sondern erlaubt zudem das Referenzieren von Laufzeitdaten, die an den Kommunikationsschnittstellen der Mashup-Komponenten entstehen. So können erweiterte Anforderungen im Szenario ② realisiert werden, die Plausibilitätsprüfungen auf Anwendungsdaten als Laufzeittests ermöglichen. Das Anforderungskonzept stellt außerdem die wissenschaftliche Grundlage dar, die für das interaktive Erstellen und Berechnen von Verteilungsoptionen in einer Multi-Device-Umgebung erforderlich ist, vgl. [MM14].

Kapitel 6

Qualitätsbewusster Entwicklungs- und Nutzungsprozess

Zur Verbesserung der Prozesse für die Entwicklung und Nutzung kompositer Web-Mashups müssen die vorgestellten Modelle sowie Spezifizierungs- und Auswertungsvorgänge für Qualitätsanforderungen in typische Abläufe integriert werden, die sich aus den Szenarien ergeben. Ähnlich wie gesellschaftliche Anforderungen und Verhaltensweisen in Gesetzen geregelt werden, steuern die in dieser Arbeit vorgestellten Qualitätsanforderungen Maßnahmen bzw. *Aktionen*, die abhängig von den ebenfalls spezifizierten *Bedingungen* Anwendung finden sollen. Sind Anforderungen als abstrakte Empfehlungen festgelegt, muss sowohl über die tatsächliche Erfüllung der Bedingungen als auch über das Eintreten konkreter Aktionen einzeln und ggf. eskalativ entschieden werden, wie es auch bei den Organen der Jurisdiktion üblich ist. Dieses zeitaufwendige Vorgehen unter Berücksichtigung weiterer externer Ressourcen, die interpretierende Aufgaben im Einzelfall wahrnehmen, ist in einer qualitätsbewussten Mashup-Infrastruktur nicht praktikabel. Viele Entscheidungen – etwa über den Austausch von Komponenten bei der automatisierten Adaption und auch im Empfehlungsprozess – müssen reproduzierbar und automatisiert getroffen werden.

Soll eine automatisierte Behandlung von Anforderungen erreicht werden, so ist die formalisierte Definition von Aktionen erforderlich. Die Mashup-Plattform kann damit eigenverantwortlich Adaptionsaktionen einleiten und so die Akzeptanz von Qualitätsanforderungen sichern und sogar durchsetzen. Aktionen können in direkter Zuordnung zur Anforderung modelliert werden und treten als Folge einer Verletzung einer Anforderung – ähnlich wie Vertragsstrafen – ein. Grundsätzlich kann die Auswertung von Qualitätsanforderungen und somit die potenzielle Durchführung von Maßnahmen in unterschiedlicher Art und Weise ausgelöst werden. Die im vorhergehenden Kapitel vorgestellten Events beinhalten dafür zum einen explizite Nutzerinteraktionen und zum anderen implizite Trigger. Diese können entweder in Reaktion auf die Änderung bestimmter Kontexteigenschaften auftreten oder werden proaktiv, beispielsweise durch intervallgesteuerte Abfragen, generiert.

Zunächst beschreibt dieses Kapitel einen *integrierten Prozess der Nutzung und Entwicklung* kompositer Web-Mashups. Auf dieser Grundlage werden die *Erweiterungen und Verbesserungen* vorgestellt, die die Arbeit mit Qualitätsanforderungen bietet. Die beschriebenen Prozesse beziehen dabei die hier entstandene Infrastruktur und die bereits vorgestellten Modelle ein. Anschließend werden die im Rahmen der Anforderungen eingeführten Aktionen, die die Anwendungsadaption und auch die Empfehlung ermöglichen, näher behandelt. Ausgehend von einem *Referenzmodell typischer Aktionen*, das die Leistungsfähigkeit des Konzeptes zeigt, stehen vor allem Strategien zur *Konfliktbehandlung* und der *Aktionslebenszyklus* im Fokus.

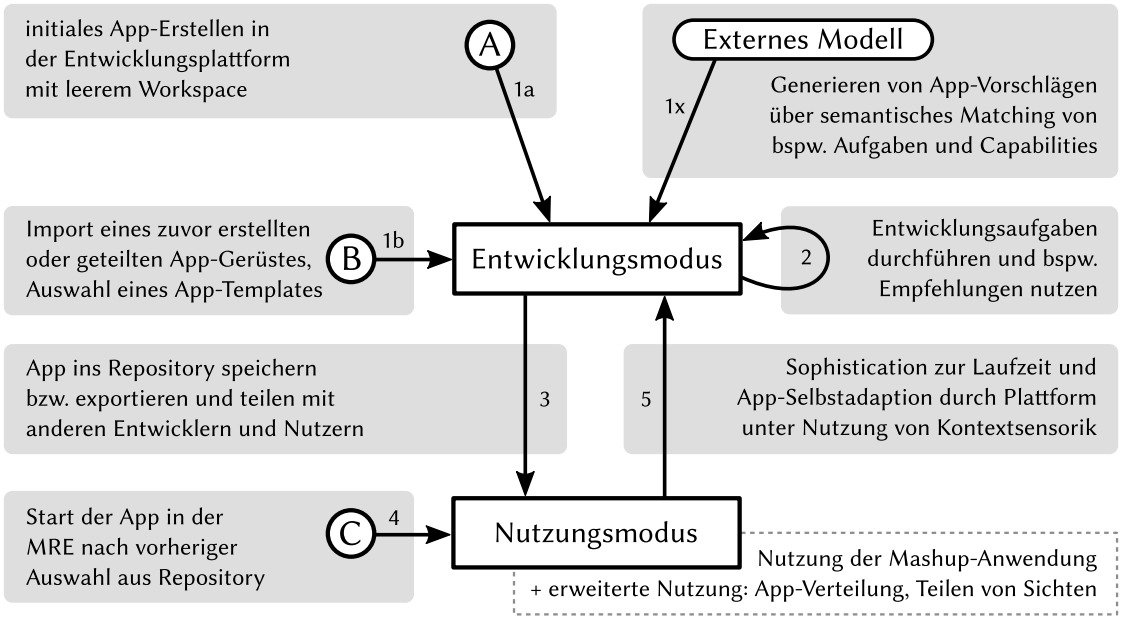

Abbildung 6.1: Entwicklungs- und Nutzungsprozess für komposite Web-Mashups

6.1 Entwicklung und Nutzung von Web-Mashups mit Qualitätsanforderungen

Unter Berücksichtigung der zugrundeliegenden Mashup-Plattform und ihrer typischen Nutzungsformen, die sich insbesondere aus den in Unterabschnitt 2.2.2 vorgestellten Anwendungsszenarien ergeben, wird in diesem Abschnitt zunächst ein grundlegender Entwicklungs- und Nutzungsprozess für komposite Web-Mashups beschrieben. Dieser bildet die Basis für Verbesserungen, die sich durch die *qualitätsbewusste* Entwicklung und Nutzung mit Anforderungen ergeben. Insgesamt wird eine Erweiterung der miteinander verbundenen Prozesse vorgestellt, die die iterative Weiterentwicklung (Live-Sophistication) unterstützen. Das verbesserte Entwicklungsvorgehen wird schließlich aus Sicht der Szenarien diskutiert und bewertet.

6.1.1 Entwicklungsprozess für komposite Web-Mashups

Ausgehend von der in Abschnitt 5.2 vorgestellten Architektur wird nun ein grundlegender Entwicklungs- und Nutzungsprozess für Mashups beschrieben. Abbildung 6.1 gibt einen Überblick zu den Zusammenhängen der wichtigsten Entwicklungs- und Nutzungsvorgänge. Dort sind insbesondere die verschiedenen Einstiegspunkte gekennzeichnet, die sich im Zuge der Rollenbetrachtung in den Szenarien – siehe Abschnitt 2.3 – voneinander abgrenzen.

Um an einer Anwendung in der Mashup-Plattform zu entwickeln, werden grundsätzlich drei Einstiegspunkte unterschieden. Ohne jede Vorlage wird mit (A) gestartet. Beginnend mit einer leeren Arbeitsumgebung wird hier über (1a) eine Anwendung völlig neu erstellt. Steht ein Aufgabenmodell zur Verfügung, das beispielsweise wie in [Tie15] aus einem Geschäftsprozessmodell abgeleitet wurde, so lässt sich über das Generieren von Kompositionsvorschlägen mit passen-

den Mashup-Komponenten eine Vorlage für die Mashup-Anwendung mitsamt den Kandidaten für Komponenten erzeugen (1x). Diese Möglichkeit findet grundsätzlich auch bei anderen Arten *externer Modelle* Anwendung, die in einem vorgelagerten Prozess das Erstellen von Kompositionsmodellen erlauben. Unter der Annahme, dass bereits Modelle für Anwendungsgerüste oder fertige Anwendungen existieren, die zuvor von anderen Entwicklern im Repository verfügbar gemacht wurden, so ist der Einstieg (1b) mit (B) als Ausgangspunkt möglich. Nach dem Durchlaufen eines dieser drei Pfade im Entwicklungsprozess befindet sich die in Betracht stehende Anwendung im *Entwicklungsmodus*. Die Entwicklungsplattform kann nun zu ihrer grundlegenden Entwicklung bzw. zur Verbesserung genutzt werden.

Entwicklungsaufgaben an der Anwendung bzw. am Kompositionsmodell werden im Wesentlichen mit Schritt ② abgedeckt. Die Plattform bietet hierfür Funktionen an, um in der Komposition Komponenten hinzuzufügen, zu entfernen oder gegen geeignetere auszutauschen. Außerdem stehen grundsätzlich alle Belange des Kompositionsmodells – wie etwa das Layout und vor allem die anwendungsinternen Kommunikationsbeziehungen – zur Bearbeitung frei. Nach ggf. iterativer Verfeinerung des Anwendungsgerüstes über mehrmaliges Durchlaufen des Vorganges wird das Mashup nun im Schritt ③ im Repository abgelegt und kann je nach Situation mit anderen Nutzern geteilt werden, die es in Schritt (1b) erneut laden und anschließend an ihre Bedürfnisse anpassen können. Neben der Ausführung einer selbst entwickelten oder modifizierten Anwendung besteht mit Schritt ④ über Ausgangspunkt (C) die grundlegende Möglichkeit der Nutzung einer komplett fertig entwickelten Anwendung, die ebenfalls im Repository gefunden werden kann. Nach Schritt ③ bzw. ④ befindet sich die Anwendung im *Nutzungsmodus*. Sie ist damit zur Ausführung innerhalb der Laufzeitumgebung bereit.

Schritt ⑤ führt die iterative Live-Sophistication als Transition in den Entwicklungsprozess ein. Außerdem werden damit Aktionen der Selbstadaption, beispielsweise durch das Auslösen bestimmter Aktionen über Kontextsensorik, abgedeckt. Die *Aufteilung in Modi* sowie die Übergänge ③ und ⑤ zwischen ihnen sind *konzeptioneller Natur*. In der Umsetzung wird hingegen eine integrierte Entwicklung und Nutzung durch nahtlose Werkzeugintegration angestrebt. Eine automatische Adaption, die ausgehend vom Nutzungsmodus einen Komponentenaustausch auslöst, würde beispielsweise den *Roundtrip* der Transitionen ⑤, ② und ③ auslösen. Der Nutzer bekäme je nach Nutzungssituation eine Benachrichtigung über den erfolgten Austausch über Awareness-Funktionen der Mashup-Plattform. Dieser Roundtrip bedient Szenario ② für die Überwachung von Qualitätsanforderungen zu Laufzeit, wofür anschließend die infrastrukturelle Erweiterung vorgestellt wird. Neben der durch die Anwendungslogik vorgegebenen Nutzung der kompositen Web-Mashup-Anwendung kann auch erweiterte Laufzeitinfrastruktur in Anspruch genommen werden, etwa durch das Hinzufügen der in der Laufzeitumgebung ausgeführten Anwendung zu einem verteilten Anwendungskontext [MM14] oder das Teilen von Anwendungsfunktionalität bzw. einzelner Sichten und Berechtigungen in kollaborativen Szenarien [BRM13].

6.1.2 Verbesserung von Entwicklungsaufgaben und Ausführung durch Qualitätsanforderungen

Aus den Anwendungsszenarien und den darauf aufbauenden konzeptionellen Betrachtungen ergeben sich an mehreren Stellen im beschriebenen Prozess Notwendigkeiten zur Verbesserung. Die wichtigsten innerhalb dieser Arbeit entstandenen Aktivitäten zur Verbesserung im Prozess

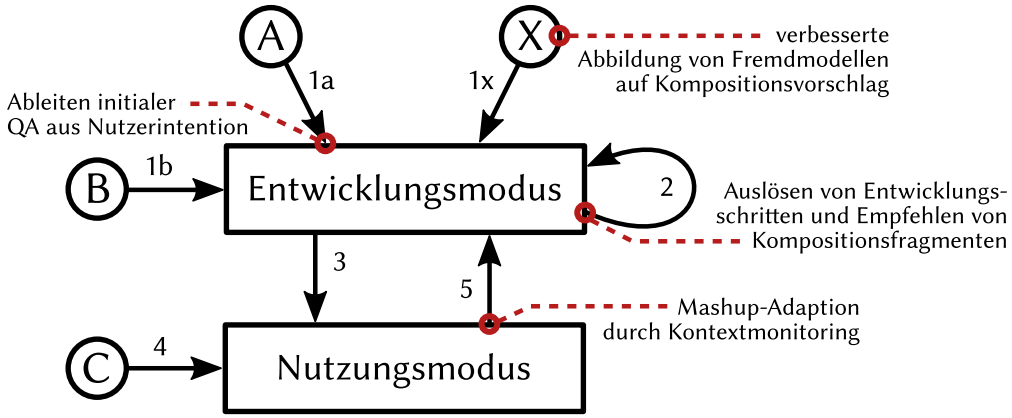

Abbildung 6.2: Anknüpfungspunkte bei der Verbesserung von Entwicklungs- und Nutzungsprozessen kompositer Web-Mashups durch Qualitätsanforderungen

sind in Abbildung 6.2 gekennzeichnet. Dafür wird der Prozess mit den Schritten aus Abbildung 6.1 ohne die zusätzlichen Erklärungen in verkürzter Form mit zusätzlichen Annotationen (kleine rote Kreise) genutzt. Eine offensichtliche Hilfe für den Nutzer der Mashup-Plattform, der von Neuem eine Anwendung erstellen möchte, ist das Ableiten initialer Qualitätsanforderungen aus seiner Intention im Schritt (1a). Die infrastrukturelle Entsprechung ist der Anforderungsassistent, der den Einstieg in die Entwicklung erleichtert, indem er unter Berücksichtigung des Nutzermodells – etwa über die Nutzung favorisierter Anforderungen – das Festlegen von Anforderungen ermöglicht. Diese Brücke zwischen Nutzerintention und formal definierter Anforderung tritt auch an anderen Stellen im Entwicklungsprozess auf. Die Abkürzung über das Auswählen beliebter Qualitätsanforderungen sowie das Bearbeiten und Speichern dieser wird in Abbildung 5.6 näher beschrieben.

Ein anderer Ansatz ist das Generieren bzw. das Vorschlagen kompositer Anwendungen aus Fremdmodellen in (1x), mit denen die hier vorgestellte Plattform nicht direkt arbeitet. Dabei handelt es sich beispielsweise um Aufgabenmodelle, die ihre Anforderungen als Aufgaben auf Fähigkeiten (englisch: *capabilities*) abbilden, um eine Anwendung oder zumindest Teile davon zu finden, die die gewünschte Aufgabe adäquat umsetzen. Die Qualitätseigenschaften aus dieser Arbeit kann das Anforderungsmodell ergänzen und somit den Prozess der Abbildung von Aufgabenbäumen auf komposite Mashup-Anwendungen präzisieren, wie es in [Tie15] vorgeschlagen wird. Alternativ erfolgt in diesem Schritt auch ein Import aus anderen Mashup-Modellen, die bereits in anderen Plattformen genutzt werden. Neben der initialen Empfehlung unter Nutzung des Anforderungsassistenten werden dessen Fähigkeiten iterativ bei der Weiterentwicklung benötigt. Dies erfolgt in vielerlei Ausprägungen in Schritt (2). Im *Überwachungsmodus* steht die automatische Adaption in Schritt (5) als zentrale Neuerung im Umgang mit Qualitätsanforderungen im Fokus. Diese wird erst durch zentrale Konzepte dieser Arbeit, wie etwa die maschinenlesbare Modellierung von Qualitätsanforderungen, die automatisierte Auswertbarkeit durch Evaluatoren und die Kopplung an deterministische Ereignisse sowie das Zuordnen von Aktionen ermöglicht. Gleichzeitig wird hierdurch und die Anbindung des Kontextmonitorings vielseitiges Potenzial für neuartige Anwendungsfälle eröffnet. Vorgelagert zu beispielsweise Ausgangspunkt (C) können in einem separaten Anwendungsbrowser die Aggregationsfunktionen

auf Anwendungsebene genutzt werden, die eine zusätzliche Abstraktionsschicht auf die Wertebelegung der Qualitätseigenschaften innerhalb der untersuchten Anwendungen bieten. Das Konzept dazu wird in Unterabschnitt 4.5.4 vorgestellt. In diesem Browser ausgewählte Anwendungen können dann mit einem Klick direkt in der Laufzeitumgebung ausgeführt werden. Anschließend steht der Weg aus dem Nutzungsmodus zur weiteren Anpassung über die Transition ⑤ in den Entwicklungsmodus frei.

Während im Überwachungsmodus eher ein binäres Auswertungsergebnis erwünscht ist, das über die Ausführung von Aktionen entscheidet, kommt es im Empfehlungsmodus auf eine Abstufung an, die das Bilden einer *Reihenfolge* erlaubt. Der Empfehlung von Kompositionsfragmenten liegen neben den Nutzeranforderungen, die den Bezug zu Qualitätsanforderungen herstellen, noch weitere Parameter zugrunde. So sind bei der Erweiterung einer bereits in ihrer Ausführung befindlichen Anwendung die Kopplungsmöglichkeiten in die Bewertung einzubeziehen. Diese können aus den Schnittstellenbestandteilen der Kompositionsfragmente bzw. aus den zusätzlichen Annotationen der Capabilitys extrahiert werden. Eine vertiefte Betrachtung weiterer Charakteristika des Empfehlungsprozesses beschreibt [Rad+12]. Auch eine Vorfilterung von Kandidaten während der Empfehlung und eine anschließende Dominanzbewertung, vgl. [RBM16], kann das Erlebnis während der Empfehlung verbessern. Für die Szenarien dieser Arbeit ist das textbasierte Filtern über den Titel, die Beschreibung und die annotierten Schlüsselwörter der Kompositionsfragmente und das anschließende Anwenden der Vergleichsprozesse über die Qualitätsanforderungen ausreichend.

Da viele Anforderungen an die in dieser Arbeit betrachteten Anwendungen letztlich von durch Menschen ausgefüllten Rollen stammen, ist es wichtig zu wissen, dass der Großteil der tatsächlichen Nutzeranforderungen zunächst verborgen bleibt. Die nach dem *Eisbergmodell* verborgenen Anforderungen müssen durch geeignete Techniken aus dem Unbewussten hervorgeholt werden [RZ74]. Das damit verbundene *Pareto-Prinzip* geht von einer 80/20-Verteilung verborgener und bewusster Anforderungen aus. Durch möglichst kleine Iterationszyklen und die Integration des Entwicklungs- und Nutzungsmodus wird dieser Herausforderung im beschriebenen Prozess begegnet. Ziel ist es außerdem, durch einen hohen Grad an Automatisierung bei der nutzergetriebenen Entwicklung, das Einbeziehen aller wichtigen Parameter, die unter der Oberfläche den unteren Teil des Eisberges ausmachen, zu fördern.

6.1.3 Qualitätsbewusste Entwicklung in den Anwendungsszenarien

Die vorgestellten Verbesserungen in der integrierten Entwicklung und Nutzung von Anwendungen in der Mashup-Plattform tragen zur verbesserten Durchführung in den Anwendungsszenarien aus Unterabschnitt 2.2.2 bei. Die konkrete Verortung der Maßnahmen und Prinzipien in den Szenarien wird nun detailliert vorgestellt.

① Live-Sophistication

Mit einer bestimmten Problemstellung als Intention für die Lösung eines individuellen Anwendungsproblems tritt der Nutzer Anton ohne Programmiererfahrung an die Mashup-Plattform heran. Eine bestimmte Aufgabe soll in einer Anwendung gelöst werden. Anton meldet sich auf dem Gerät seiner Wahl an der webbasierten Mashup-Plattform CRUISE an. Zuerst wird der Startbildschirm präsentiert, in der sowohl der Anforderungsassistent im Empfehlungsmodus

als auch weitere Optionen zur Auswahl bestehender Anwendungen zu sehen sind. Die bereitgestellte Auswahl berücksichtigt zuletzt vom Nutzer verwendete Anwendungen sowie populäre Anwendungen aus der Community und für den Nutzer freigegebene Anwendungen von Kollaborationspartnern. Wird nicht der direkte Start einer Anwendung in der Laufzeitumgebung genutzt, so kann die iterative Auswahl über den Anforderungsassistenten verwendet werden. Dort besteht die Möglichkeit, eine *Anwendungsdomäne* bzw. ein *Thema* auszuwählen. In dieser Arbeit kommen beispielhaft die Domänen *Tourismus* (Reiseplanung, Sehenswürdigkeiten, Mobilität, Wetter), *Finanzen* (persönliches Banking, Wertpapiere und Wirtschaftsnachrichten) sowie *Unterhaltung* (Videos, Social Media) vor. Durch die Domänenauswahl kann eine erste inhaltliche Einschränkung getroffen werden, indem nach domänenspezifischen Capabilitys gefiltert wird, die gegen die Annotationen der Kompositionsfragmente verglichen werden können. Die Verfeinerung der Auswahl einer Anwendungsdomäne stellt ein verfeinertes Matching bzw. eine Auswahl von Capabilitys dar. Im Beispielszenario würde hier die Domäne Tourismus ausgewählt werden. Unter der Annahme, dass in der gewählten Domäne sehr viele Kompositionsfragmente angeboten werden, kann nun die Verfeinerung der Anforderungen unter Zuhilfenahme der Qualitätseigenschaften erfolgen. Die für Anton zunächst einfachste Verfahrensweise ist die Auswahl eines passenden Qualitätsprofils, vgl. Unterabschnitt 5.4.4. Passend zum Szenario wäre hier das Profil »Premiumbewertung« geeignet. Durch das gespeicherte Profil seiner Qualitätsanforderungen aus vergangenen Sitzungen, kann dem Nutzer bereits eine Vorauswahl präsentiert werden, die er nur noch zu bestätigen braucht. Diese favorisierten Anforderungen können sowohl die inhaltlichen Ziele (Goals) inklusive der Anwendungsdomäne als auch Qualitätsanforderungen sein, die im einfachsten Fall durch Qualitätsprofile abstrahiert wurden. Bei jeder Anpassung der Anforderungen wird die Ansicht der empfohlenen Kompositionsfragmente aktualisiert, sodass iterativ eine genauere Abbildung der Nutzerintention entsteht. Findet Anton eine hinreichend genaue Entsprechung unter den Kandidaten, kann er diese innerhalb der Laufzeitumgebung sofort integrieren und dann die Anwendung nutzen. Er führt damit implizit den Übergang ③ aus dem Entwicklungsprozess aus. Der Anforderungsassistent steht auch bei bereits laufenden Anwendungen, die beispielsweise durch das zuvor integrierte Kompositionsfragment zustande gekommen sind, zur erneuten Empfehlung bereit. Antons Intention im Beispielszenario ist eine Anwendungserweiterung, die er mit Unterstützung des Assistenten vornimmt. Im Unterschied zur ersten Empfehlung steht die Anwendungsdomäne bereits fest. Zusätzlich kann bei der Menge angebotener Kompositionsfragmente nach der Passgenauigkeit bezüglich der schon integrierten Anwendungsbestandteile gefiltert werden. Sind die Qualitätsanforderungen, die für die zu findende Erweiterung gelten sollen, nicht als Qualitätsprofil repräsentiert, kann über den Editor für Bedingungen die entsprechende Bedingung unter Nutzung der Qualitätseigenschaften und der anderen Konzepte wie Concerns und Fuzzy-Terme erstellt werden. Im Beispiel reicht Anton die simple Auswahl des Qualitätsprofils nicht aus, sodass er mit dem Condition-Editor weitere Qualitätseigenschaften, wie den Preis einer Komponente, einbezieht. Durch das Angeben des Fuzzy-Terms »kostenlos« und der Angabe entsprechender Junktoren kann er die Qualitätsanforderungen der Plattform wie gewünscht mitteilen. Die Komposition mehrerer Bedingungen erfolgt über die Tags in der Anforderungsleiste. Dort können sowohl bestimmte Teilbedingungen ausgewählt als auch die Struktur des Bedingungsbaumes angepasst werden.

② Überwachung von Anforderungen und Plausibilitätstest

Wird zum soeben beschriebenen Empfehlungsszenario als Rolle ein fortgeschrittener Nutzer oder Experte für Anforderungen hinzugezogen, der Erfahrungen in der Programmierung bzw. Modellierung hat, ergeben sich zwei weitere nützliche Szenarien in der Überwachung von Qualitätsanforderungen bzw. in der Plausibilitätsüberprüfung. Soll die Einhaltung von Bedingungen innerhalb von Qualitätsanforderungen zur Laufzeit überwacht werden, so muss zunächst eine lauffähige Mashup-Anwendung zur Verfügung stehen. Ob diese durch einen professionellen Anwendungsentwickler erstellt wurde und über das Repository verbreitet wurde oder durch Live-Sophistication iterativ durch einen Nutzer zustande gekommen ist, spielt keine Rolle. Wichtig ist, dass die Anwendung mit zur Laufzeit überprüfbaren Qualitätsanforderungen versehen wurde. Dabei müssen diese Anforderungen im Gegensatz zu denen, die im Empfehlungsmodus genutzt werden, explizite Angaben über Events und Actions enthalten. Sind die Anwendungen noch nicht mit den erforderlichen Anforderungen ausgestattet, muss sie ein Experte für Qualitätsanforderungen nachrüsten. Alle dafür notwendigen Angaben können im Überwachungsmodus des Anforderungsassistenten vorgenommen werden. Anders als im Empfehlungsmodus kann dieser in zwei *Scopes* (englisch für Gültigkeitsbereiche) erreicht werden. Sollen die festzulegenden Anforderungen im Kontext der gesamten Anwendung – also für alle enthaltenen Kompositionsfragmente – gelten, wird der Assistent über das Anwendungsmenü in der Nutzerschnittstelle der Laufzeitumgebung aktiviert. Den direkten Einstieg im Scope einer bestimmten Komponente bietet eine kleine Schaltfläche in ihrer jeweiligen Titelleiste. Das Festlegen der kompositen Bedingungen erfolgt analog zu den Bedingungen im Empfehlungsmodus über den passenden Editor. Hinzu kommt hier das Festlegen der Events und Actions. Das Event wird der Anforderung zugeordnet, indem die entsprechende Schaltfläche in der Repräsentation der Anforderung in der Anforderungsleiste gewählt wird. Anschließend kann das Event im Editor für Events ausgewählt und ggf. konfiguriert werden. Die Auswahl orientiert sich dabei an der Menge von Events aus Unterabschnitt 5.3.4. Analog dazu erfolgt die Auswahl und Konfiguration von Actions im Editor für Actions.

Ist die Anforderung komplett spezifiziert, kann sie direkt im Überwachungsmodus des Assistenten aktiviert werden. Mit diesem Schritt erfolgt die Registrierung eines Listeners für das angegebene Event in der Laufzeitumgebung. Dabei können mehrere Anforderungen gleichzeitig – auch auf denselben Eigenschaften und Events – aktiv sein. Konflikte sind auf dieser Betrachtungsebene aus Plattformsicht zunächst ausgeschlossen. Unter bestimmten Umständen können jedoch bereits hier vorbeugend Konflikte, die beim Ausführen von Aktionen entstehen können, ausgeschlossen werden. Eine Konfliktmatrix für Aktionen wird in Unterabschnitt 6.2.3 detailliert betrachtet. Die Awareness über eingetretene Aktionen gegenüber dem Nutzer der Anwendung liegt in der Verantwortung der Aktion. Hierzu stellt die Laufzeitplattform mehrere Mechanismen, wie das Einbinden in einen Empfehlungsprozess oder das Setzen von Benachrichtigungen, bereit. Je nach Intention der ausgeführten Aktion der Qualitätsanforderung kann eine explizite Benachrichtigung auch unerwünscht sein. Die Ausprägung der Anforderungen lässt somit flexibel die vollautomatisierte Durchführung bestimmter Adaptionen aber auch die halbautomatische Reaktion auf sich ändernde Bedingungen – beispielsweise über das erneute Anstoßen einer Empfehlung und die nutzergetriebene Auswahl – zu. Unter bestimmten Umständen kann nach ein- oder mehrmaligem Eintreten eines Events die Aktivierung der Anforderung beendet werden. Dies ist typischerweise der Fall, wenn einmalige Zeitpunkte erreicht werden.

Qualitätsanforderungen dieser Art, die innerhalb eines Überwachungsprozesses während der Laufzeit von Mashup-Anwendungen aktiv sind, können neben ihrer Entstehung durch Anforderungsexperten, die der professionellen Anwendungsentwicklung nahestehen, auch durch versierte Nutzer nachträglich ergänzt werden. Ein wichtiger Anwendungsfall ist dabei das Festlegen von Testdaten, mit denen Plausibilitätsüberprüfungen der in der Anwendung vorhandenen Funktionalität bzw. der Daten, die an Kommunikationsschnittstellen auftreten, durchgeführt werden können. Sowohl das Einhalten von Wertebereichen – vor allem bei numerischen Daten – als auch das Überprüfen der strukturellen Validität sind mit dem Konzept der kompositen Bedingungen in den Qualitätsanforderungen dieser Arbeit durchführbar. Wichtig dabei ist, dass innerhalb von Bedingungen neben den Qualitätseigenschaften aus dem Referenzmodell auch Elemente der Schnittstellen der Kompositionsfragmente angeboten werden. Konkret bedeutet das, dass Wertebelegungen des Zustandes von Mashup-Komponenten auf Vergleichswerte und gegenüber Fuzzy-Sets geprüft werden können. Hierfür werden neben den Typen der Qualitätseigenschaften auch die Propertys aller aktuell in der Anwendung integrierten Mashup-Komponenten angeboten. Damit kann beispielsweise sichergestellt werden, dass sich die aktuelle Lufttemperatur für einen bestimmten Ort in einer Wetterkomponente in einem bestimmten Wertebereich bewegt. Die Werteänderungen der so adressierten Propertys können analog zu den sich ändernden Kontextwerten der Qualitätseigenschaften ermittelt werden, die über die Sensorik zur Verfügung gestellt werden.

③ Automatische Interaktion zwischen Geräten

Das Festlegen und Vorhalten von Qualitätsanforderungen ohne direkte menschliche Interaktion wird durch die umfassende Anforderungsmodellierung als RDF-Daten und die serialisierten Darstellungen, wie beispielsweise TURTLE, ermöglicht. Dadurch können Anforderungen sowohl in Kontextmodellen von Geräten, Nutzern, Plattformen und Kompositionsfragmenten vorgehalten als auch zwischen ihnen ausgetauscht werden. Kommt beispielsweise ein Zusammentreffen zwischen einer Mashup-Anwendung und einem Gerät zustande, indem die Anwendung in der Laufzeitplattform diesem Gerät zugeordnet wurde, so können Anforderungen in der Anwendung mitgeliefert und bei der Ausführung an das Zielgerät gestellt werden. Offenbart sich bei den gestellten Bedingungen ein Problem, wird über zugeordnete Aktionen reagiert, indem etwa ein alternatives Gerät oder eine für das Gerät geeignete Zweitanwendung bzw. eine alternative Konfiguration der Komponenten ausgewählt wird. Beispiele für Anforderungen zwischen den in einer Mashup-Plattform agierenden Trägern von Qualitätseigenschaften sind auch in Abbildung 5.2 zu sehen.

6.1.4 Diskussion von Entwicklungsparadigmen für Web-Mashups

In der Mashup-Plattform CRUISE wird grundsätzlich ein aufbauendes bzw. *additives* Entwicklungsparadigma verfolgt. Ausgehend von einem leeren Arbeitsbereich oder einer existierenden Anwendung wird die gewünschte Funktionalität ergänzt. Kommunikationsbeziehungen zwischen integrierten Anwendungsbestandteilen können empfohlen und als Datenflusskanäle eingefügt werden. Der Ausgangszustand ist jedoch, dass die Mashup-Komponenten nicht gekoppelt sind. Grundsätzlich kann der Nutzer jedoch gut nachvollziehen, welche Anwendungsbestandteile und Verbindungen zwischen ihnen existieren, weil er sie explizit oder durch Emp-

fehlung hinzugefügt hat. Außerdem steht ihm die nachträgliche Kontrolle über einen Entwicklungsmodus offen, der bestehende Kommunikationskanäle visualisiert. Die Entwicklungswerkzeuge der Mashup-Plattform lassen auch das Entfernen von Kompositionsfragmenten und Kommunikationsbeziehungen zu. Ein anderes Paradigma ist bei der Entwicklung mit der Mashup-Plattform aus dem Forschungsprojekt *OMELETTE*, vgl. [Chu+13], verbunden. Hier werden automatisch alle Kommunikationsbeziehungen, die die ausgewählten Anwendungsbestandteile anbieten, von der Laufzeitumgebung eingerichtet. Nicht erwünschte Funktionalität muss der Nutzer manuell entfernen. Das Vorgehen ist somit *subtraktiv* bezüglich der Schaffung von Anwendungslogik durch das Einrichten von Kommunikationsbeziehungen. Vor allem bei Mashup-Komponenten mit sehr generischen Kommunikationsschnittstellen kann bei OMELETTE viel von dieser unerwünschten Funktionalität entstehen. Der Entwicklungsaufwand ist somit im Vergleich zu CRUISE hoch. Im in dieser Arbeit beschriebenen Ansatz zum Festlegen und Auswerten von Qualitätsanforderungen ist die Komposition von Anwendungsfunktionalität auf das positive Angeben von Intentionen und damit gewünschter Funktionalität ausgerichtet. Daher und in Verbindung mit vorgeschlagener Kommunikation ist ein additiver Ansatz zur Mashup-Komposition geeigneter, weil damit eine Reduktion des Entwicklungsaufwandes verbunden ist. Auf den Überwachungsmodus ist der subtraktive Ansatz eher übertragbar, da er auf unveränderten Anwendungen und festen Scopes arbeitet.

Zum Lebenszyklus von Qualitätsanforderungen wird im Rahmen von Aktionen später Stellung genommen. Aus Prozesssicht ist jedoch abzugrenzen, was passieren soll, wenn im Entwicklungsschritt ② eine Komponente hinzugefügt wird und zuvor über das Anwendungsmenü eine Qualitätsanforderung angelegt wurde, die für alle Komponenten in der Anwendung gelten soll. Hierbei sind drei Optionen der Behandlung bzw. der vorgegebenen Geltung möglich. Entweder sie gilt auch für die neue Komponente, sie gilt nicht für die neue Komponente oder der Nutzer wird bezüglich der Geltung der Anforderung immer gefragt. Eine sinnvolle Vorgabeoption kann nicht pauschal gelten, jedoch ist sie als Konfiguration einer konkreten Instanz der Mashup-Plattform beispielsweise als Unternehmensrichtlinie möglich.

Das strikte Filtern nach Schnittstellen und deren Kompatibilität ist nur sinnvoll, wenn ein *explorativ*es Entwicklungsparadigma unterstellt wird. Dieses Vorgehen der direkten Weiterentwicklung ausgehend vom vorhandenen Zustand wird in vielen Fällen bei der Arbeit mit der CRUISE-Plattform unterstellt. Soll hingegen ein *antizipierendes* Paradigma angewendet werden, so kann das semantische Matching hinderlich sein. Nach diesem Paradigma soll die Zielanwendung beispielsweise eine linear verkettete Kommunikation zwischen den Komponenten A und B sowie zwischen B und C realisieren. Zunächst ist Komponente A integriert. Der Nutzer möchte als nächstes Komponente C integrieren und erst dann Komponente B dazwischen einfügen. Wenn die Komponenten A und C jedoch semantisch nicht kompatibel sind, ist die Empfehlung kontraproduktiv, weil dem Nutzer keine Ergebnisse nach seiner Entwicklungsintention angeboten werden. Semantisches Matching auf Capabilitys basierend auf der Kompatibilität von Schnittstellen kann also auch irreführend innerhalb von Empfehlungsprozessen sein. Ein weiterführendes Empfehlungssystem auf Basis von Qualitätsanforderungen für zu integrierende Kompositionsfragmente stärkt somit die Leistungsfähigkeit der Mashup-Plattform in besonderen Fällen. Sollen beide Paradigmen gewürdigt werden, ist es sinnvoll, ein intendiertes Paradigma aus dem Kontext zu gewinnen bzw. vom Nutzer oder aus historischen Instanzen der Entwicklungsprozesse ableiten zu lassen. Eine Unsicherheit bleibt jedoch immer bestehen, wenn das Paradigma spontan geändert werden soll.

6.2 Aktionen im Kontext von Qualitätsanforderungen

Als wichtigster Bestandteil einer Qualitätsanforderung in Bezug auf deren Durchsetzbarkeit gilt die zugeordnete Aktion. Sie bietet eine planbare Maßnahme, um auf die Verletzung der in der Anforderung hinterlegten Bedingung bzw. des Bedingungsbaumes angemessen zu reagieren. Dabei können sowohl vollautomatische Aktionen wie das Austauschen von Anwendungsbestandteilen, das empfehlungsbasierte Reagieren im Zusammenspiel mit der Mashup-Plattform als auch das simple Benachrichtigen eines Nutzers oder Entwicklers abgebildet werden. Abhängig von der Ausrichtung der gewählten Aktion entscheidet sich der weitere Ablauf innerhalb der in diesem Kapitel bereits vorgestellten Entwicklungsprozesse. Das Zuordnen von Aktionen zu Qualitätsanforderungen wird im Metamodell – vgl. Unterabschnitt 5.3.1 – erklärt. Dieses Kapitel geht zunächst auf Details dieser Beziehung und die Integration der Aktionen in die Mashup-Infrastruktur ein. Anschließend werden typische Aktionen, die in der Entwicklung und Überwachung von Mashup-Anwendungen genutzt werden, strukturiert in einem Referenzmodell vorgestellt. Für viele Aktionen ist eine Parametrisierung notwendig, die ebenfalls erklärt wird. Schließlich wird auf den Lebenszyklus von Aktionen und mögliche Konflikte bei ihrer Aktivierung und Nutzung eingegangen.

6.2.1 Entstehung und strukturelle Einordnung

Aktionen als operativer Teil von Qualitätsanforderungen wurden zuerst in [RM12] vorgestellt. Die CRUISE-Plattform bietet konzeptionell das Adaptivitätsmodell und insbesondere die Adaptionstechniken für komposite Web-Anwendungen an, vgl. [Pie12, Abschnitt 5.3]. Dort wird auch das regelbasierte ECA-Prinzip genutzt. Die Benennung lehnt sich allerdings an die Begriffe der Aspektorientierung – beispielsweise *Advice* und *Pointcut* – an und es wird nur mit einzelnen domänenspezifischen Beispielen, wie dem Austausch von Anwendungsbestandteilen bei niedrigem Batteriestand, gearbeitet. Die komplexen Entwicklungsaufgaben einer Mashup-Plattform, die auch für die Live-Sophistication und automatisiert auswertbare Qualitätsanforderungen genutzt werden soll, verlangen nach einer differenzierten Betrachtung – auch des Referenzmodells der Adaptionsaktionen.

Wie in Abbildung 6.3 als Ausschnitt des Metamodells für Qualitätsanforderungen, vgl. Abbildung 5.3, gezeigt, ordnen sich Aktionen als Teil eines ECA-Musters ein. Mit diesem Tripel aus Event Condition Action werden Regeln nach dem Vorbild der ereignisgetriebenen Architektur (EDA) umgesetzt. Das *Event* legt in dieser Regel zunächst fest, zu welchen Gelegenheiten die Qualitätsanforderung in den Fokus der Beobachtung gerät. Tritt der spezifizierte Moment – ggf. wiederholt – ein, so wird die als *Condition* beschriebene Bedingung über die Evaluatoren ausgewertet. Abhängig vom Auswertungsergebnis wird schließlich das Auslösen der als *Action* festgelegten Maßnahmen durch den Qualitätsmonitor veranlasst. Unter Nutzung mehrerer Aktionen pro Anforderung können auch umfangreiche Adaptionen vorgenommen werden. Die Parameter für jede Aktion werden durch die hohe Heterogenität nicht im Metamodell, sondern im Referenzmodell auch auf RDF-Ebene spezifiziert.

6.2.2 Referenzmodell möglicher Arten durchführbarer Aktionen

Die vielfältigen Einsatzmöglichkeiten durchsetzbarer Qualitätsanforderungen, die exemplarisch durch die Anwendungsszenarien ausgedrückt werden, bieten ein breites Potenzial für verschie-

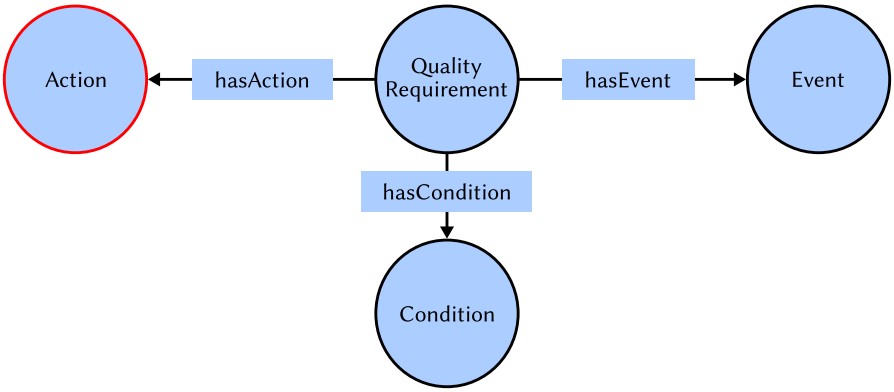

Abbildung 6.3: *Action* als Teil des ECA-Musters im Metamodell für Qualitätsanforderungen

dene Arten von *Aktionen*. Sie erweitern hauptsächlich die *Adaptivität* der mit der Mashup-Plattform entstehenden Anwendungen. Aktionen können von unterschiedlichen Entitäten in der Infrastruktur bereitgestellt bzw. erbracht werden. Dabei stellen bestimmte Ausprägungen den Nutzer in den Mittelpunkt und beteiligen diesen, beispielsweise beim Einholen von Feedback in problematischen Situationen. Mit einem ähnlich hohen Maß an Kontrolle kommen Aktionen aus, die Vorschläge unterbreiten und in einem Empfehlungsprozess halbautomatisch für eine Lösung sorgen. Im Gegensatz dazu ist in bestimmten Situationen auch die vollautomatische Reaktion erwünscht, für die ebenfalls eine Palette möglicher Maßnahmen existiert. Die folgende Übersicht enthält eine strukturierte Auflistung typischer Aktionen, die für die Nutzung und Entwicklung kompositer Mashups geeignet sind. Dabei wird auch auf die Konfiguration mit benötigten Parametern eingegangen.

Benachrichtigung und Feedback

Diese Gruppe von Aktionen verursacht den größten Grad an Nutzerinteraktion. Dabei ist es sowohl möglich, den unmittelbar mit der Anwendung agierenden Nutzer informativ bzw. steuernd einzubeziehen, als auch die Unterstützung von Experten in bestimmten Situation einzufordern, indem eine kollaborative Anwendungssitzung aufgebaut wird.

- Über eine **Benachrichtigung** kann der Nutzer einer Anwendung über einen bestimmten Ausgang der Anforderungsauswertung informiert werden. Praktisch wird dies über ein Popup oder ein Benachrichtigungsfenster bzw. das Highlighten einer bestimmten Komponente bzw. einer Region innerhalb des UI dieser Komponente bzw. der Anwendung realisiert. Diese Aktion zählt also als *Awareness-Funktion*, die über eine Nachricht parametrisiert wird. Vor allem in Verbindung mit Szenarien der Plausibilitätsüberprüfung können dem Nutzer die gewünschten Warnungen zur Kenntnis gegeben werden.
- Das **Auffordern des Nutzers zur Eingabe von Feedback** liefert durch die Kommentarmöglichkeit in Textform einen Rückkanal vom Nutzer zum Entwickler der genutzten Kompositionsfragmente bzw. anderen Nutzern des Repositorys. Dieses Vorgehen bietet sich bei nicht automatisch behebbaren Fehlerfällen oder aus Mangel an Alternativen an.

Abseits von Testläufen, in denen ausführliches Feedback wichtig ist, können zum Bilden von Statistiken oder als Empfehlungshilfe für andere Nutzer neben dem Feedback als Freitext auch strukturierte Angaben, wie Fünf-Sterne-Ratings oder Radiobuttons, zugelassen werden.

- Ist keine automatische Maßnahme vorgesehen und soll dem Nutzer auch keine eigenverantwortliche Entscheidung im Problemfall zugemutet werden, kann das **Hinzuziehen eines Experten** veranlasst werden. Die Anwendungssitzung wird dabei erweitert, indem beispielsweise ein anderer Nutzer oder Support-Mitarbeiter zur synchronen Kollaboration eingeladen wird. Voraussetzung hierfür ist, dass die Plattform das Freigeben und die damit verbundenen Funktionen der Kollaboration und Awareness anbietet. Zu beachten ist hierbei, dass ggf. ein zeitlicher Bruch entsteht, wenn der Experte nicht sofort zur Verfügung steht. Alternativ kann auch die asynchrone Beratung zu einem bestimmten Sachverhalt als Aktion etabliert werden.

Veränderung des Anwendungsmodells

Sollen Veränderungen in der Anwendung vorgenommen werden, so geschieht das durch die Veränderung des Anwendungsmodells (MCM) des kompositen Mashups. Die konkreten Optionen erstrecken sich vom Austausch einzelner Mashup-Komponenten bis zur Veränderung von Kommunikationsbeziehungen oder dem Anpassen des Layouts und der Verteilung.

- Um der Anwendung eine **Mashup-Komponente hinzuzufügen**, die Funktionalität ergänzt, welche durch die Anforderung als fehlend identifiziert wurde, muss der *Komponentenbezeichner* als Parameter angegeben werden. Dabei ist es auch möglich, eine weitere Instanz einer Mashup-Komponente zu integrieren, die bereits in der Anwendung vorhanden ist, beispielsweise zur gleichzeitigen Visualisierung des Wetters an zwei verschiedenen Orten. Die Unterscheidung erfolgt durch die bei der Integration vergebenen Instanzbezeichner. Hat die hinzugefügte Komponente ein UI, so müssen die Auswirkungen auf die Kommunikation, das Layout und ggf. den Screenflow berücksichtigt werden. In den meisten Fällen kümmert sich die Laufzeitumgebung im Rahmen eines dynamischen Layouts darum bzw. stellt günstige Optionen über den Vorschlagsmechanismus zur Auswahl.

- Das **Entfernen einer Komponente** erfolgt unter Angabe des *Instanzbezeichners* als Parameter. Kommunikationsbeziehungen werden grundsätzlich automatisch von der zu entfernenden Komponente entkoppelt, indem die beteiligten Publisher und Subscriber in den relevanten Kommunikationskanälen ausgetragen werden. Fehlt anschließend eine gewünschte Kopplung, so muss diese über den Entwicklungsmodus oder über eine Aktion ergänzt werden. Die Behandlung des Layouts und des Screenflows richtet sich nach den UI-Adaptionsregeln der Laufzeitumgebung, analog zu der Aktion des Hinzufügens.

- Das **Ersetzen** einer Komponente kann als Kombination des Hinzufügens und Entfernens abgebildet werden. Die Konsequenzen bezüglich der benötigten Parameter und die Auswirkungen auf die Kommunikation, das Layout und den Screenflow ergeben sich aus den einzelnen Aktionen. Bei einem 1:1-Tausch wird der *Instanzbezeichner* der alten und der *Komponentenbezeichner* der neuen Komponente angegeben.

- Besitzt das Mashup mehrere Sichten (englisch: *views*), so kann durch eine Aktion das **Wechseln von Views** veranlasst werden. Solche Aktionen stellen eine automatisierte Bedienung innerhalb der vorgesehenen Grenzen der Anwendung dar, ohne ihr Anwendungsmodell zu verändern. Als Parameter muss hier der *Bezeichner des Ziel-Layouts* angegeben werden.

- Wird die Zusammensetzung der kompositen Anwendung beibehalten, so kann durch das **Verändern der Kommunikationskanäle** im Kommunikationsmodell die Anwendungslogik beeinflusst werden. Sowohl das Hinzufügen und Entfernen von Publishern und Subscribern in vorhandenen Kommunikationskanälen als auch das Erstellen und Entfernen von Kommunikationskanälen ist möglich.

- Eine auf Visualisierung und Interaktion ausgerichtete Aktion ist das **Anpassen des Layouts**. Hier wird die Anordnung und Abmessung der sichtbaren Mashup-Komponenten verändert. Komponenten ohne UI sind hiervon nicht betroffen. Mehrere Layouts sind über Event-gesteuerte Screenflows verbunden. Durch das **Anpassen des Screenflows** kann die Interaktionsstruktur und Übersichtlichkeit verändert werden. Vor allem durch Platzbeschränkungen bei kleinen Bildschirmen ist die intelligente Aufteilung der visuellen Inhalte auf verschiedene Layouts mittels Screenflows hilfreich.

- Lässt die Mashup-Plattform das verteilte Ausführen von Anwendungen auf mehreren Geräten zu, ist als Aktion auch das **Verändern der Verteilung der Anwendungsbestandteile** über das Verteilungsmodell erlaubt. Damit lassen sich unter anderem Komponenten zwischen Geräten migrieren, beispielsweise bei niedrigem Akkustand. Konzeptionelle Grundlagen der verteilten Ausführung kompositer Mashups werden in [MM14] vorgestellt.

Implementierung und Selbstadaption von Anwendungsbestandteilen

Betrifft der Geltungsbereich einer Aktion nur bestimmte Charakteristika einer Komponente, so kann entweder eine angebotene Schnittstelle zur Selbstadaption genutzt oder ein anderes Binding für die Implementierung gewählt werden.

- Komponenten bieten Schnittstellen zur Adaption an, die für das **Auslösen komponenteninterner Selbstadaption** geeignet sind. Durch eine dazu passende Adaptionsaktion können diese Schnittstellen gezielt angesprochen werden. Der Black-Box-Charakter der Mashup-Komponenten wird somit bezüglich der Adaptionsflexibilität des Mashups abgemildert. Propertys und Operations können als *vorhandene Strukturen der Komponentenschnittstellen* für diese Adaptionsmöglichkeit als Parameter genutzt werden.

- Der Komponentendeskriptor bietet über das flexible Zuordnen verschiedener Komponentenimplementierungen den **Austausch von Bindings** an. Diese können im Rahmen einer Aktion gegeneinander ersetzt werden, indem *das neue Binding* als Parameter angegeben wird. Beim Austausch des Bindings muss die integrierte Komponente in ihrem Container neu geladen werden. Der durch Propertys erfasste Zustand wird aufgrund der identischen Schnittstelle nach dem Integrieren des neuen Bindings neu gesetzt.

Laufzeitplattform

Als erweiterte Funktionen der Laufzeitumgebung für Mashups können *Empfehlungen* genutzt werden. Der Nutzer trägt hier mit seiner Auswahl vorberechneter Optionen zur Erfüllung der Aktion in begrenztem Maße bei. Auch das Migrieren in einer Multi-Device-Umgebung oder das Ausführen einfacher Systemfunktionen (Logging) sind als Aktionen verfügbar.

- Für das Ergänzen von Funktionalität kann durch das vorgeschlagene Integrieren einer Mashup-Komponente das **Empfehlen von Kompositionsfragmenten** als Aktion genutzt werden. Alternativ steht das Empfehlungssystem auch für eine alternative Kommunikation und andere elementare Aktionen bereit. Die für die Empfehlung genutzten Parameter richten sich nach der konkreten Situation. Sie können neben den ausgeführten Nutzereingaben und den integrierten Komponenten auch die Daten des Nutzerprofils, insbesondere vordefinierte Anforderungen aus dem Nutzerprofil, berücksichtigen.
- Während die Migration von Anwendungskomponenten zwischen mehreren Geräten die Verteilung im Anwendungsmodell verändert, kann durch die **Migration von Komponenten zwischen Client und Server** der Ausführungskontext optimiert werden, indem beispielsweise energieintensive Berechnungen in Komponenten ohne UI auf einen serverseitigen Ausführungskontext verschoben werden. Als Parameter werden hier die *Bezeichner der zu migrierenden Komponente und Zielumgebung* benötigt.
- Mit der Aktion **Logging** werden Protokolle angefertigt, die das direkte Nutzererlebnis nicht beeinflussen. Auch für das Überwachen und nachträgliche Auswerten von Logs für Plausibilitätsregeln ist das Logging geeignet, vor allem in Verbindung mit automatisierten Auswertungswerkzeugen für das Data- und Process-Mining. Als Parameter wird der Aktion die *zu loggende Nachricht* beigefügt.
- Mit der **Wahl einer alternativen Anwendungskonfiguration** können vorgefertigte Anwendungen geladen werden, die die aktuell ausgeführte Anwendung ersetzen. Dies bietet sich vor allem an, wenn weitreichende Probleme in der alten Anwendung identifiziert werden. Der Parameter ist hierbei der *Bezeichner der zu ladenden Anwendung*. Optional wird noch angegeben, ob die neue Anwendung die alte ersetzen oder in einem weiteren Tab der Laufzeitumgebung geladen werden soll.
- Als Hilfsmittel für Entwickler oder auch für einen Kioskmodus steht als Aktion das **Umschalten zwischen dem Live-Modus und den beiden Entwicklermodi** (englische Bezeichnungen: *capability mode* und *professional mode*) zur Verfügung. Der *Zielmodus* wird dabei als Parameter angegeben.
- Eine zusätzliche Hilfe für die Nutzung von Mashups ist der **Erklärungsmodus** (englisch: *explanation mode*). Dieser kann über eine Aktion aktiviert werden und kommt ohne Parameter aus.
- Durch die Wahl eines anderen von der Plattform angebotenen **Qualitätsprofils** können Qualitätsanforderungen selbst modifiziert werden. Ein Qualitätsprofil beinhaltet eine *Menge von Qualitätsanforderungen*. Die Anforderungen werden vorgefertigt geladen und als Parameter angegeben.

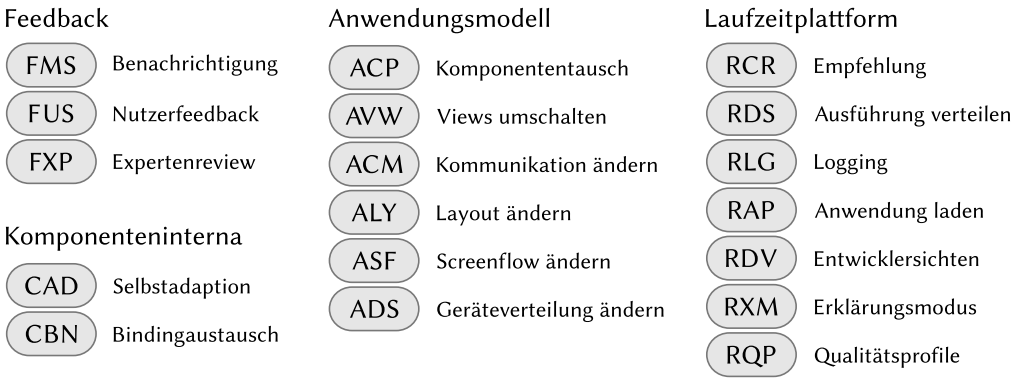

Abbildung 6.4: Gruppen von Aktionen im Referenzmodell mit Kürzeln

Die vorgestellten Aktionen und ihre Gruppen sind in Abbildung 6.4 im Überblick dargestellt. Jede Aktion des Referenzmodells erhält dabei ein Kürzel, das jeweils in der anschließenden Betrachtung zu Konflikten zwischen Aktionen verwendet wird.

6.2.3 Konflikte zwischen Adaptionsaktionen

Abbildung 6.5 zeigt das Potenzial für Konflikte zwischen Adaptionsaktionen, wobei das *zeitnahe Eintreten jeweils zweier Aktionen* angenommen wird. Beim Ausführen selbst können nur in sehr seltenen Fällen Probleme aus Sicht der Laufzeitumgebung auftreten. Die Konfliktarten sind in der Abbildung farbig zusammengefasst, vgl. Legende. Hellgrüne Zellen stehen dabei für Schwierigkeiten, die sich für Nutzer im UI ergeben bzw. eine *visuelle Überlastung* verursachen. Gelbe Zellen beschreiben einen *Kontextverlust* für die eine Aktion, der durch das Anwenden einer anderen ausgelöst wird, wie eine wegfallende Referenz zu einem zu bearbeitenden Objekt. Orange Zellen sind mögliche *Feedback-Schleifen*. Rote Zellen können einen *ungültigen Zustand in der Plattform* herbeiführen. Zum zeitnahen Zusammentreffen mehrerer Aktionen kommt es aus bestimmten Gründen. Zunächst werden mehrere Aktionen – die innerhalb einer Anforderung demselben Event zugeordnet sind – gemeinsam ausgelöst, sofern die Erfüllung der Bedingungen gegeben ist. Außerdem kann es zum Zusammentreffen von Aktionen mehrerer Anforderungen kommen, wenn diese entweder ebenfalls mit demselben Event verbunden sind oder zufällig zeitnah aufgrund verschiedener Events auftreten. Die sich daraus ergebenden Paarungen sind auf das Zusammentreffen von drei oder mehr Aktionen übertragbar.

Aktionen mit besonders geringem Konfliktpotenzial und zwei Sonderfälle sind in der Matrix aus Platzgründen nicht dargestellt. Dazu zählt das Umschalten von Views im Screenflow (AVW), das Ändern des Screenflows (ASF) und das Logging (RLG). Außerdem ist das Laden einer neuen Anwendung (RAP) und das Anwenden von Qualitätsprofilen (RQP) nicht berücksichtigt. Beim Laden einer neuen Anwendung, die die aktuelle Anwendung ersetzt, kommt es in fast allen Fällen zu einem Kontextverlust. Geschieht das Laden nicht-desktruktiv in einem weiteren Tab, so ist das Konfliktpotenzial minimal und vor allem auf die Awareness gegenüber dem Nutzer beschränkt. Das Anwenden von Qualitätsprofilen ist so vielfältig, dass sich nur schwer allgemeine Aussagen zum Interaktionsverhalten mit anderen Aktionen treffen lassen. In vielen Fällen besteht das Risiko eines Kontextverlustes.

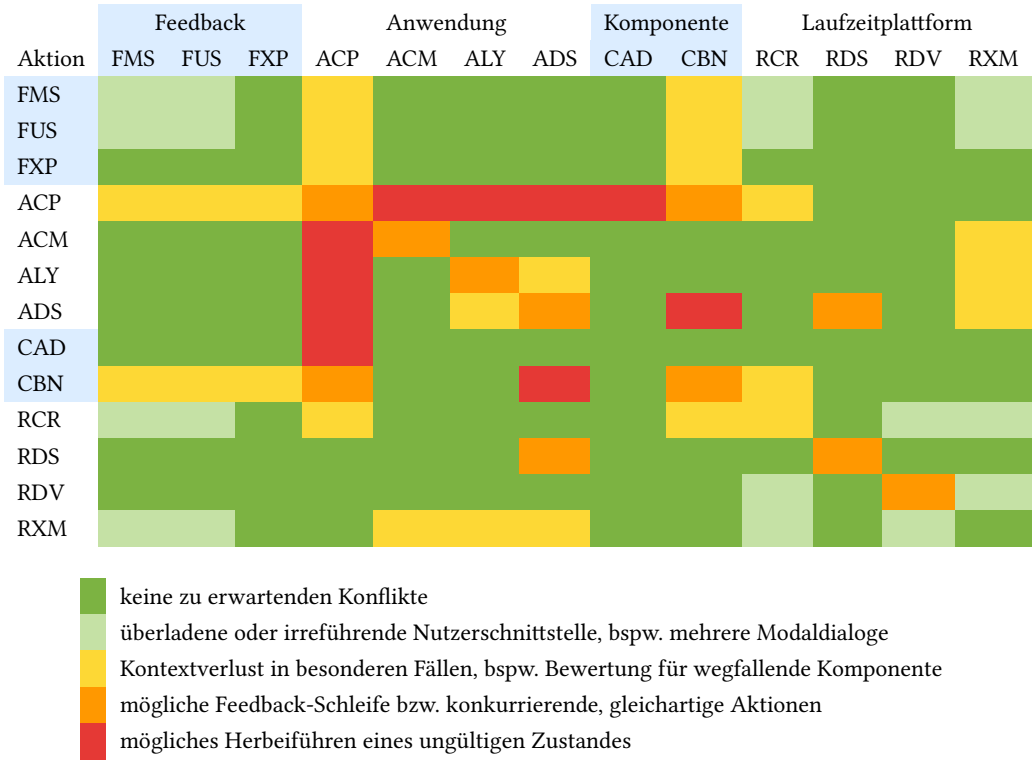

Abbildung 6.5: Konfliktmatrix zwischen jeweils zwei zeitnah eintretenden Aktionen

Oben links in der Matrix ist zu sehen, dass sich Benachrichtigungen (FMS) und die Aufforderung zur Eingabe von Nutzer-Feedback (FUS) visuell stören können. Dies kann durch geschickte Anordnung im UI oder durch Beschränkungen für Modaldialoge vermieden oder zumindest eingeschränkt werden. Die Aufforderung zur Hilfestellung durch einen weiteren Experten (FXP) verläuft weitgehend asynchron und hat wenig Einfluss auf die unmittelbare Nutzerinteraktion bzw. das Geschehen der laufenden Anwendung und ihrer Bestandteile. In der Aktionsgruppe der Veränderung des Anwendungsmodells haben Modifikationen im *Conceptual Model* die weitreichendsten Auswirkungen. Bei der zugehörigen Aktion (ACP) wird festgelegt, welche Komponenten entfernt werden und welche zur Anwendung hinzugefügt werden sollen. Das größte Problem hierbei ist, dass sich viele andere Bestandteile des Anwendungsmodells auf die verwendeten Komponenten beziehen, beispielsweise im Kommunikationsmodell und im Layout. Es kann hierbei – zumindest temporär – zu ungültigen Zuständen kommen, vor allem, wenn Komponenten entfernt werden. Steht ACP mit sich selbst oder mit dem Austausch eines Bindings, also der Komponentenimplementierung, (CBN) in Interaktion, so können auch Feedback-Schleifen auftreten, wenn eine Anforderung die Integration der neuen Komponente mit einer weiteren Aktion rückgängig macht und dies als Input für eine Anforderung dient. Außer im Erklärungsmodus und im gerade beschriebenen Fall der Veränderung der Komponenten in der Anwendung haben Modifikationen des Kommunikationsmodells (ACM) kaum störendes Potenzial bei der Interaktion mit anderen Aktionen, außer mit sich selbst. Ähnliches gilt für das

Verändern des Layouts (ALY). Dort gilt allerdings die Besonderheit, dass sich beim Verteilen auf andere Geräte (ADS) das Layout zwangsläufig ändert. Dabei können als Gegenmaßnahmen automatisierte Layout-Techniken als Vorgabe, wie etwa ein lineares *Flow-Layout*, einspringen. Das Auslösen der Selbstadaption einer Komponente über das Ansprechen der Kommunikationsschnittstellen (CAD) ist gegenüber anderen Aktionen verträglich, außer es handelt sich um ACP. Aktionen, die sich mit dem Inhalt der Komponente beschäftigen, können auf Nutzerebene in Konflikt geraten, wenn das Binding ausgetauscht (CBN) wird. Ein besonderes Problem kann außerdem auftreten, wenn ein ersetztes Binding auf andere Clients verteilt (ADS) werden soll. In diesem Fall muss sichergestellt sein, dass das Binding auch für die Zielplattform geeignet ist. Eine Empfehlung, die für eine Komponente ausgesprochen wurde (RCR), verliert ihren Wert, wenn die Komponente im selben Moment bereits hinzugefügt wurde oder der restliche Anwendungskontext durch ACP verloren geht. Die Verteilung von Komponenten auf eine andere Ausführungsumgebung (RDS) hat hauptsächlich darauf Einfluss, wenn sie gleichzeitig auf ein anderes Client-Gerät migriert werden sollen (ADS). Unproblematisch ist das Umschalten des Entwicklungsmodus (RDV) als Aktion, die hauptsächlich mit sich selbst konkurriert. Beim Aktivieren des Erklärungsmodus (RXM) kann es in der durch das Kompositionsmodell aufgebauten Anwendungslogik zu Kontextverlusten kommen.

Das Betrachten der Paarungen möglicher Konflikte in Abhängigkeit von der Art der ausgeführten Aktionen zeigt, dass aus Sicht der Laufzeitumgebung kaum Konfliktfälle entstehen können. Hauptsächlich treten sie bei *invalidierten Referenzen im Anwendungsmodell* auf. Aus Nutzersicht sind zusätzliche Fälle zu betrachten, die zu Kontextverlusten und damit verbundenen visuellen Problemen – beispielsweise bei den angebotenen Optionen während einer Empfehlung – führen. Zudem können Awareness- und Benachrichtigungsfunktionen überladene Nutzerschnittstellen dadurch verursachen, dass *zu viele Informationen in Modaldialogen gleichzeitig angezeigt* werden. Eine *Konfliktprävention* ist möglich, indem durch eine *pessimistische Nebenläufigkeitsbehandlung* das Erstellen von Bedingungen bei gleichzeitiger Existenz von Anforderungen mit Konfliktpotenzial bereits im Editor bzw. beim Import verhindert wird.

6.2.4 Lebenszyklus von Aktionen und Anforderungen

Qualitätsanforderungen bestimmen die Ausführungspfade im kombinierten Prozess der Entwicklung und Nutzung, vor allem bei ihrer Aktivierung in Überwachungsszenarien und während der Empfehlung. Ermöglicht wird dies durch den Einsatz von Aktionen. Dabei unterliegen sie einem Lebenszyklus, der das Entstehen, Empfehlen, Verarbeiten, Vorhalten und Entsorgen regelt. Aktionen entstehen mit der Qualitätsanforderung oder werden ihr nachträglich zugeordnet. Die Entstehungszeitpunkte sind somit bis zur Zuordnung zwischen Aktion und Anforderung unabhängig voneinander. Soll eine Aktion für den Überwachungsmodus festgelegt werden, so wird sie über den entsprechenden Editor angelegt, vgl. Unterabschnitt 5.4.2. Neben der Zuordnung erfolgt im Editor auch gleich die Aktivierung, sofern bereits ein Event zugeordnet wurde. Nicht nur das nutzergetriebene Erstellen, sondern auch das Importieren bestehender oder generierter Anforderungen mit zugeordneten Aktionen ist möglich, indem diese als serialisiertes Instanzmodell über das Nutzerprofil oder als Teil der Anwendung zur Verfügung stehen. Auch während eine bereits zugeordnete Aktion für ein Event aktiviert ist, können weitere Aktionen zugeordnet werden. Ebenso möglich ist das Entfernen aktiver Aktionen über den Editor im Anforderungsassistenten. Sollen Änderungen an zugeordneten Aktionen vorgenom-

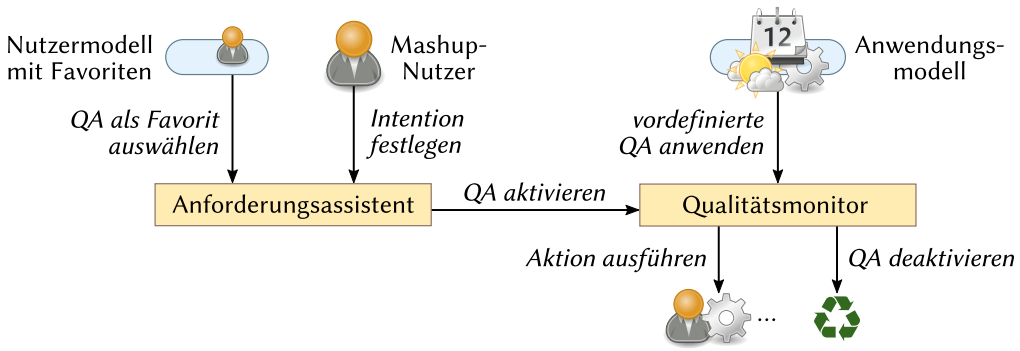

Abbildung 6.6: Lebenszyklus von Aktionen in der Mashup-Plattform

men werden, so ist es möglich, die betreffende Aktion in der Anforderungsleiste auszuwählen. Als Folge dessen werden in den Editoren für Events und Actions die bisherigen Werte geladen. Sie stehen dann zum Ändern bereit. Die verschiedenen Typen von Aktionen im Referenzmodell unterscheiden sich dabei im Umfang ihrer Konfigurierbarkeit. So muss beispielsweise beim Austausch von Mashup-Komponenten angegeben werden, welche Komponente ausgetauscht wird und welche an ihre Stelle tritt. Möglicherweise werden Teile dieser konfigurierbaren Parameter auch ausgelagert und ein bestimmter Dienst innerhalb der Infrastruktur ist für die Auswahl passenden Ersatzes verantwortlich. Andere Arten von Aktionen, beispielsweise das Aktivieren des Erklärungsmodus, kommen ohne zusätzliche Konfigurationsparameter aus.

In bestimmten Fällen können Events nicht erneut auftreten, wie beispielsweise bei einmaligem Erreichen eines vorgegebenen Zeitpunktes. Die Anforderungsauswertung kann dann nicht erneut erfolgen und somit die Aktion nicht wieder ausgeführt werden. In diesen Fällen kann der Qualitätsmonitor in der Laufzeitumgebung entscheiden, dass die Anforderung deaktiviert wird. Sie steht dann jedoch weiterhin über das Nutzerprofil zur Auswahl bereit, sofern sie dort als Favorit hinzugefügt wurde. Bei implizit festgelegten Aktionen, wie sie im Empfehlungsmodus auftreten, ist der Lebenszyklus noch höher automatisiert. Sowohl das Zuordnen als auch das Entfernen nach der erfolgten Empfehlungsaktion sind vollautomatische Vorgänge.

Abbildung 6.6 fasst die Lebensphasen von Aktionen zusammen. Im oberen Teil der Abbildung sind die Quellen von Aktionen zu sehen, aus denen auch die Art des Festlegens und die Stelle der Interaktion mit der Mashup-Infrastruktur hervorgeht. Wird die Anforderung mit ihren zugeordneten Aktionen bereits in einer Anwendung mitgeliefert, so erfolgt ihre Aktivierung ohne Nutzerbeteiligung beim Integrieren der Anwendung in die Laufzeitumgebung. *Halbautomatisch* können Anforderungen durch das Laden von Favoriten aus dem Nutzermodell genutzt werden. Ist der Nutzer mit den bereits zugeordneten Aktionen zufrieden, so kann er sie durch das Akzeptieren im Anforderungsassistent bestätigen und aktivieren. *Völlig nutzergetrieben* gelangen die Intentionen über festzulegende Aktionen im entsprechenden Editor in die Mashup-Plattform. Unabhängig von der Anforderungsquelle werden die Qualitätsanforderung mit ihrer Aktivierung dem Qualitätsmonitor zugeführt. Wird dort festgestellt, dass das zugeordnete Event nicht mehr erreichbar ist, erfolgt die Deaktivierung der Anforderung, weil die zugeordnete Aktion nicht mehr ausgeführt werden kann. Die wichtigere Aufgabe ist jedoch das Auslösen der Aktionsausführung, die je nach Typ die Laufzeitumgebung und ggf. den Nutzer einbezieht.

Kapitel 7

Validierung und Implementierung

Dieses Kapitel beschreibt zunächst die Methodik der Validierung, die für die verschiedenen Ergebnisse dieser Arbeit eingesetzt wurde. Zu den durchgeführten Maßnahmen gehört die umfassende Implementierung der in den konzeptionellen Kapiteln vorgestellten Modelle, Funktionalitäten und Prozesse als Bestandteile der Referenzarchitektur. Basierend auf den Forschungszielen aus Abschnitt 1.3 wird die Eignung der Konzepte für die Anwendungsszenarien aus Unterabschnitt 2.2.2 gezeigt. Zuerst werden die für diese Arbeit erstellten Instanzmodelle und Implementierungen der Kompositionsfragmente vorgestellt. Anschließend erfolgt die Beschreibung der wichtigsten Artefakte der implementierten Referenzarchitektur zur Verwaltung von Qualitätseigenschaften und zur Auswertung von Qualitätsanforderungen. Über verschiedene Rollen der Entwicklungs- und Laufzeit erstrecken sich die Einsatzmöglichkeiten der danach vorgestellten Werkzeugimplementierungen. Dazu gehören vor allem der Anforderungsassistent als Teil der Laufzeitumgebung, der Fuzzy-Set-Editor und die Visualisierungswerkzeuge für Kompositionsfragmente. Die genannten Implementierungsarbeiten sind die wesentliche Grundlage für eine Reihe von durchgeführten Tests, die das restliche Kapitel zur Validierung der Forschungsergebnisse bestimmen. Zunächst zeigt die Auswertung einer Nutzerstudie zum Festlegen anpassbarer Qualitätsanforderungen die *Rezeption und mögliche Grenzen* des im Rahmen der Live-Sophistication gewünschten Vorgehens auf. Weitere Validierungsergebnisse beinhalten Messergebnisse zur Performance der wichtigsten automatisiert durchgeführten Prozesse, wie der Anforderungsauswertung. Außerdem werden das testgetriebene Vorgehen und die Ergebnisse von Expertenbefragungen zu Details der Nutzerschnittstelle beschrieben. Die wichtigsten Erkenntnisse des Validierungskapitels und die aufgrund der Ergebnisse vorgenommenen Verbesserungen werden danach kurz zusammengefasst. Sie liefern die Basis für die Diskussion im letzten Kapitel, das eine umfassende Bewertung der Ergebnisse dieser Arbeit vornimmt.

7.1 Validierungsmethodik

Im Kontext der Validierungsmethodik wird zwischen grundlegenden Ergebnissen, wie Modellen, dem Entwicklungs- und Nutzungsprozess und Software-Artefakten, die bereits eine Validierungsfunktion dieser grundlegenden Ergebnisse erfüllen, unterschieden. Dazu gehört die Implementierung der Referenzinfrastruktur, der Werkzeuge und das Bereitstellen von Beispieldaten. Ein Zwischenergebnis sind dabei UI-Konzepte, die die Vorlage für die Nutzerführung innerhalb der Werkzeuge darstellen. In ihnen und innerhalb von Experten- bzw. Nutzerstudien zeigt sich, welche der grundlegenden Konzepte die Zielgruppe tatsächlich in gewünschter Weise beim Erfüllen ihrer Aufgaben unterstützen. Die verschiedenen Arten der Forschungsergebnisse erfordern angepasste Methoden der Validierung. Ziel der Validierungsprozesse ist dabei das

Überprüfen der Eignung einer Software oder eines Systems für einen bestimmten Einsatzzweck. Die durch iterative Verbesserung gewonnenen Ergebnisse dienen dem Sicherstellen bzw. der Optimierung der Effektivität der Artefakte der Mashup-Plattform als Softwareprodukt.

Durch die *Implementierung einer größeren Anzahl von Kompositionsfragmenten als Beispieldaten* in verschiedenen Anwendungsdomänen können mehrere Aspekte validiert werden. Einerseits wird so die Eignung der implementierten Software-Plattform für eine größere Anzahl an Kompositionsfragmenten erprobt, andererseits werden damit die Modellierungsgrundlage und die Erweiterungen in den Beschreibungen der Kompositionsfragmente geprüft. Daneben kann durch die Verteilung der implementierten Beispieldaten auf mehrere Anwendungsdomänen die Projektion der Konzepte auf Anwendungsfälle außerhalb der definierten Szenarien gezeigt werden. Details und Statistiken zu den implementierten Kompositionsfragmenten liefert Abschnitt 7.3.

Implizit – das heißt *by design* – validiert werden kann zudem die *Interoperabilität* in der Modellierung der Mashup-Infrastruktur durch die Nutzung von standardisierten Technologien. Das betrifft vor allem die in RDF und OWL umgesetzten Modelle für Kompositionsfragmente und verschiedene Kontexte. Das einfache Einbinden von Drittmodellen, vgl. Tabelle 4.2, die Serialisierung, die Adressierung sowie die Nutzung von verbreiteten Frameworks zur Verarbeitung wurden in dieser Arbeit bereits beschrieben. Weitere Synergien offenbaren sich in der erweiterten Nutzung von Reasoning-Fähigkeiten und der Datentyp-Mediation, die durch die Integration in parallele Forschungsarbeiten zur Geltung kommt, beispielsweise [RBM16] und [MM14].

Ausgehend von den Komplexitätsbetrachtungen für potenziell zeit- bzw. speicherintensive Vorgänge kann durch *Performance-Messung und -Optimierung* eine Verbesserung bzw. genauere Einschätzung der konzipierten und implementierten Algorithmen und Prozesse erreicht werden. Dafür ist vor allem eine Betrachtung der Skalierungsgrößen, wie die Anzahl der Kompositionsfragmente, die während einer Empfehlung in Frage kommen, wichtig. Einige Ergebnisse von Performance-Benchmarks werden in Abschnitt 7.7 vorgestellt.

Sowohl die Konzepte der Modellierung als auch das UI-Konzept für die Werkzeuge und der Nutzungs- bzw. Entwicklungsprozess können auf ihre Eignung für die angestrebte Zielgruppe überprüft werden. Dabei ist zu unterscheiden, ob die Testpersonen selbst aus der Zielgruppe stammen oder die Studie mit Experten durchgeführt wird, die sich in die einzelnen Aufgabenstellungen eindenken müssen. Die Ergebnisse einer Nutzerstudie zur Benutzung des Anforderungsassistenten werden in Abschnitt 7.6 vorgestellt. Dort wird auch Stellung zu den dadurch inspirierten Verbesserungen in der Nutzerführung und der Implementierung genommen. Weitere Ergebnisse, die durch Expertenbewertungen detaillierter UI-Konzepte zustande gekommen sind, beschreibt Abschnitt 7.7.

Da die Referenzinfrastruktur und auch die genutzten Ressourcen der Mashup-Komponenten einen hohen Verteilungsgrad bezüglich ihrer Lagerung und Auslieferung im Netzwerk aufweisen, stellt die *reproduzierbare Ausführung und Erprobung von Anwendungsszenarien* eine Herausforderung dar. Zur Validierung des *konstanten Verhaltens* über die Entwicklungsphasen hinweg wurde an vielen Stellen eine *testgetriebene Entwicklung* durchgeführt. Diese beinhaltet das automatisierte Ausführen von Unit-Tests, die die gewünschten Anwendungsszenarien und den Zustand der Referenzinfrastruktur reproduzieren. Ein Beispiel für einen solchen Unit-Test zeigt Codeausschnitt E.1. Über versionierbare Referenzen auf die benötigten Web-Ressourcen, die von einem Versionsverwaltungssystem geliefert werden, wurde auch dafür gesorgt, dass die benötigten Kompositionsfragmente reproduzierbar bereitstehen. Das Deployment wird dadurch

vereinfacht, dass wesentliche Teile wie die Repositorys, die Laufzeitumgebung und der Nutzerdienst als Docker-Container[1] inklusive des Befüllens mit Testdatensätzen genutzt werden können. Die Zuordnung der Ergebnisse zu den Thesen und Zielen nimmt Abschnitt 7.8 vor.

7.2 Überblick der implementierten Infrastruktur

Abbildung 7.1 zeigt im Rahmen dieser Forschungsarbeit neu hinzugekommene und substanziell erweiterte Teile der Infrastruktur zur Entwicklung und Nutzung kompositer Web-Mashups. Dabei zeigt sich, dass der auf der nutzergetriebenen Live-Sophistication [Rüm+11] basierende Entwicklungsprozess querschneidend zu den relevanten Schichten verläuft und auf die anderen Artefakte der Mashup-Plattform zugeschnitten ist. Die Modelle zur Beschreibung von Kompositionsfragmenten SMCDL und MCM sowie ihre Entsprechungen in RDF wurden auf die Integration für Qualitätseigenschaften und -anforderungen angepasst. Zudem wurden zahlreiche neue Instanzen der Kompositionsfragmente für das Durchführen von Benchmarks und zur Umsetzung der Szenarien neu erstellt. Die dort integrierten Konzepte zur Modellierung von Qualitätseigenschaften und -anforderungen werden ausführlich in Kapitel 4 und Kapitel 5 vorgestellt. Sie durchdringen sowohl die Metamodellierung als auch die Beispieldaten als Referenzmodelle. Daneben beschreibt diese Arbeit die Teilmodelle für Aktionen und Ereignisse mit Referenzeinträgen sowie die Integration für existierende Domänenmodelle und Kontextmodelle, beispielsweise zum Speichern favorisierter Anforderungen für einen Nutzer.

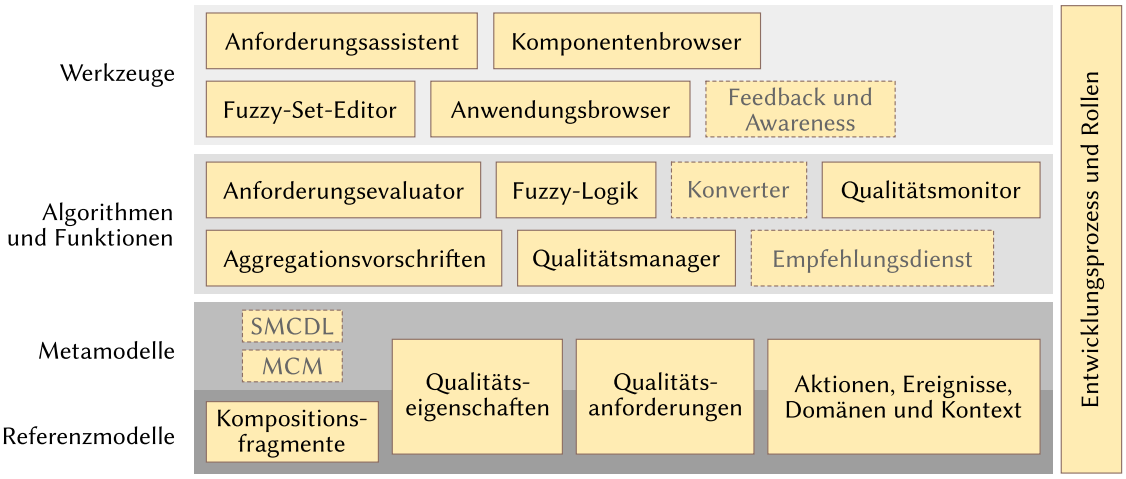

Abbildung 7.1: Wesentliche Teile der implementierten Mashup-Infrastruktur: *Neuerungen* → durchgezogene Rahmen sowie *Erweiterungen* → gestrichelte Rahmen

Aufbauend auf der Modellierung befinden sich in der nächsten Schicht neue bzw. erweiterte *Algorithmen und Funktionen* für die anforderungsgetriebene Entwicklung in der CRUISE-Plattform. Die dafür benötigten Artefakte sind hauptsächlich innerhalb der Laufzeitumgebung und im Komponenten-Repository verortet. Der *Anforderungsevaluator* (kurz: *Evaluator*) ist das Herzstück für die Auswertung von Qualitätsanforderungen. Er ist in beiden soeben genannten

[1] Docker, eine Software für das automatisierte Bereitstellen von Anwendungen in Containern: `docker.com`

Teilen der Mashup-Plattform implementiert und wird vom *Qualitätsmonitor*, der ein Teil der Laufzeitumgebung ist, aktiviert. Die Konzepte der *Fuzzy-Logik* werden innerhalb der Auswertung durch den Evaluator verwendet. *Aggregationsvorschriften* auf Anwendungsebene sind Teil des Anwendungsbrowsers. Ein grundlegender Empfehlungsprozess für Kompositionsfragmente wurde in der Kompositionsplattform CRUISE im Rahmen von [Rad+12] eingeführt und in dieser Arbeit um die Belange der Qualitätsanforderungen ergänzt. Grundlage dafür ist ein REST-basierter *Empfehlungsdienst*, der nach Vorbild des Dienstes für die Empfehlung nach funktionaler Übereinstimmung grundsätzlich neu erstellt wurde. In der genannten Arbeit von Radeck u. a. wurden zudem weitere Konzepte, auch unter Nutzung von Mustern und Synonymen, die mit annotierten Capabilitys arbeiten, parallel zu dieser Arbeit entwickelt. Das Parsen von Kompositionsfragmenten im *Konverter* wurde um die rekursive Angabe von Qualitätseigenschaften erweitert. Die Verwaltung dieser Eigenschaften erfolgt im *Qualitätsmanager*, wo auch die Aggregation von gesammelten Eigenschaftswerten erfolgt.

Die Werkzeugschicht spricht verschiedene Nutzergruppen an, die am Entwicklungs- und Nutzungsprozess beteiligt sind. Nutzergetriebene Anforderungen werden primär durch den *Anforderungsassistenten* im Empfehlungs- und Überwachungsmodus und den Anwendungsbrowser bedient. Als Unterstützung für die Entwickler, Autoren der Kompositionsfragmente und Anforderungsexperten sind der *Fuzzy-Set-Editor* und der *Komponentenbrowser* entstanden. Feedback- und Awareness-Funktionen sind als Teile der Laufzeitumgebung umgesetzt worden, die in anderen Bereichen, beispielsweise für kollaborative Anwendungssitzungen, parallel entwickelt wurden, vgl. [BRM13]. Einen zusätzlichen Einstieg über anwendungsweit aggregierte Qualitätseigenschaften bietet der *Anwendungsbrowser*. Er ermöglicht das direkte Ausführen von Mashup-Anwendungen in der CRUISE-Laufzeitumgebung.

7.3 Implementierung von Kompositionsfragmenten

Grundlage für das Anreichern von Kompositionsfragmenten mit Qualitätseigenschaften und das darauf aufbauende Festlegen von Qualitätsanforderungen ist die Umsetzung des Modells und dessen Integration in die Mashup-Plattform. Durch die etablierten Formen der Serialisierung für die gewählte Modellierung in RDF bzw. OWL fällt diese Integration sehr leicht. Die Modelle für Qualitätseigenschaften und -anforderungen wurden in TURTLE umgesetzt, das von einschlägigen Frameworks wie *Jena*, das im Component Repository (CoRe) eingesetzt wird, standardmäßig unterstützt wird. Neben der Verarbeitung der Modelle durch Bibliotheken kann bei Bedarf von den Reasoning-Fähigkeiten und von standardisierten Visualisierungsformen, vgl. VOWL, Gebrauch gemacht werden. Mit der Einführung der Qualitätsmodelle mussten auch an den existierenden Modellen für die Kompositionsfragmente Veränderungen vorgenommen werden. Das betrifft sowohl die XML-Serialisierungen, die vom Repository ausgeliefert und in der Laufzeitumgebung verarbeitet werden, als auch die RDF-Entsprechungen, die im Repository für die Verwaltung von Eigenschaften und das Durchführen der Anforderungsauswertung existieren. Details zu diesen Vorgängen werden in Abschnitt 7.4 beschrieben. Zunächst wird auf die innerhalb dieser Arbeit angereicherten und innerhalb der Referenzszenarien hinzugekommenen Mashup-Komponenten eingegangen. Anschließend folgt die Betrachtung der neuen Mashup-Anwendungen, die als Beispiele für das Durchführen nutzergetriebener Entwicklungsaufgaben und von Benchmarks in den Testfällen genutzt werden.

7.3.1 Angereicherte Mashup-Komponenten

In dieser Arbeit wurden sowohl vorhandene Mashup-Komponenten der Plattform CRUISE um Metadaten und Funktionalität bzw. Sensorik angereichert als auch neue Komponenten mit der entsprechenden Ausstattung innerhalb der Szenarien erstellt. Zu einer Mashup-Komponente gehören folgende Bestandteile: Die *SMCDL-Instanz* wird als Komponentendeskriptor im Repository registriert und referenziert *Dependencys* in den *Bindings*. Als wichtigste Dependency gilt dabei die – hier weitgehend in JavaScript realisierte – Implementierung der Komponente, die das Lebenszyklusmodell für Mashup-Komponenten umsetzt, vgl. [Pie12, Unterabschnitt 6.2.3]. Diese kann weitere Bibliotheken oder auch CSS-Stylesheets und andere Ressourcen referenzieren, die wahlweise als Dependencys im Komponentendeskriptor oder über JavaScript in das Inlineframe der Komponente geladen werden.

Für die in dieser Arbeit genutzte Mashup-Plattform CRUISE wurden 105 verschiedene Komponenten als SMCDL neu entwickelt bzw. angereichert, um sie im anforderungsgetriebenen Entwicklungsprozess als Instanzen in kompositen Anwendungen nutzen zu können. Sie beinhalten zusammen 7439 XML-Knoten. Hierbei nicht berücksichtigt sind Komponenten aus Entwicklungszweigen, die für bestimmte Aspekte der Mashup-Plattform in parallelen Forschungsarbeiten entwickelt wurden, beispielsweise zu Multi-Device-Mashups oder für die kollaborative Nutzung. Inhaltlich decken die meisten Komponenten davon die Domänen Tourismus, Wissensmanagement, Nachrichten, Finanzberatung, Office und Unterhaltung ab. Durch die Kapselung der Komponenten in Inlineframes ist das mehrfache Instanziieren einer Komponente innerhalb einer Anwendung problemlos möglich, beispielsweise wenn die Wettervorhersage für zwei verschiedene Orte parallel angezeigt werden soll. Eine Erweiterung der Laufzeitumgebung gegenüber der Implementierung aus [Pie12, Unterabschnitt 7.2.2] gewährleistet die Ausführung in *Sandboxes*. Die Komponenten können sich somit nicht ungewollt beeinflussen.

Die Anreicherung mit Qualitätseigenschaften betrifft innerhalb der SMCDL vor allem die statisch festgelegten Eigenschaftswerte. Daneben wurde für bestimmte Komponenten innerhalb der Implementierung die *Sensorinfrastruktur* für messbare Qualitätseigenschaften geschaffen, wie etwa das Messen von Dienstantwortzeiten. Die Implementierungen der Komponenten in JavaScript umfassen ohne externe Bibliotheken ca. 41 000 Source Lines of Code (SLOC). Dabei werden die Zeilen des Quellcodes ohne Leerzeilen und Kommentare gezählt. Ein erheblicher Teil davon wurde für die Erprobung der Konzepte dieser Arbeit geschaffen. Zu den Deskriptoren und den JavaScript-Ressourcen kommen vor allem noch CSS-Dateien, Grafiken und weitere Ressourcen, die unter anderem als Testdaten genutzt werden. In Abbildung 7.2 wird ein Ausschnitt von registrierten Mashup-Komponenten verschiedener Domänen gezeigt, die in einem Repository registriert sind und über eine webbasierte Ansicht durchsucht werden können.

In Zusammenhang mit der Implementierung wurde ein Template für JavaScript eingeführt, das die neue Syntax für Klassen auf Basis von ECMAScript 2015 (ES6) umsetzt und Vorgaben für den Lifecycle anbietet, siehe Codeausschnitt C.2. Die Implementierung der Beispielkomponente für die Anzeige des aktuellen Wetters für einen vorgegebenen Ort verwendet dieses Template, indem sie über `extends` von der Klasse `MashupComponent` erbt, siehe Codeausschnitt C.1. Dort ist auch der Sensor für die messbare Qualitätseigenschaft *ResponseTime* in der Methode `show` zu sehen. Die Implementierung in JavaScript kann für die Beispielkomponente schlank ausfallen, weil über `super` die Vorgabefunktionalität des Templates genutzt wird.

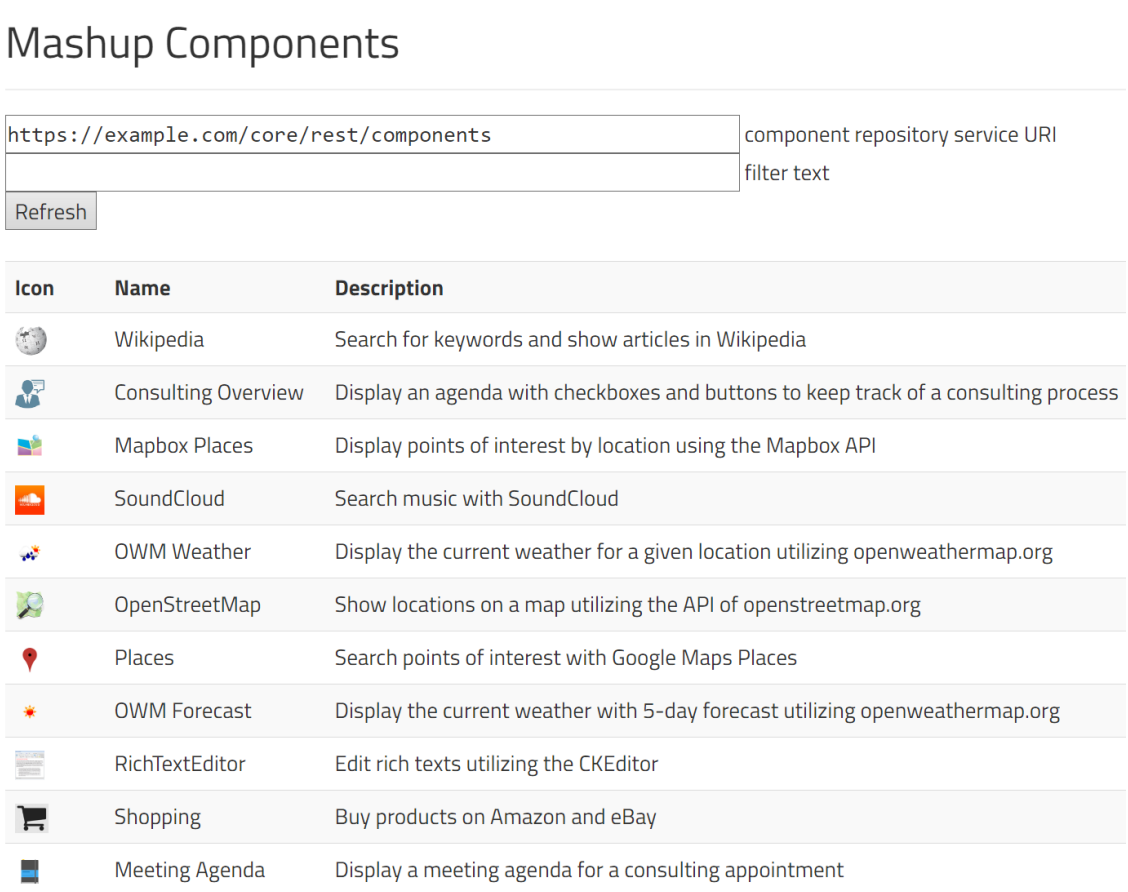

Abbildung 7.2: Beispielübersicht von Komponenten im Repository

7.3.2 Angereicherte Mashup-Anwendungen

Ein Hauptaspekt bei der Nutzung der CRUISE-Plattform ist das situative Erstellen kompositer Anwendungen über den Prozess der iterativen Live-Sophistication. Dabei sind die entstehenden Anwendungsmodelle nicht im Vorfeld festgelegt, sondern entstehen während der nutzergetriebenen Entwicklung. Der Raum möglicher Anwendungen ist somit von den verfügbaren Mashup-Komponenten und deren Interaktionsfähigkeiten abhängig. Daneben existiert auch die Option, vorgefertigte Anwendungen zu laden und in der Laufzeitumgebung ohne zusätzliche Entwicklungstätigkeit auszuführen. Auf diese Weise lässt sich mit minimalem Aufwand an den Entwicklungsergebnissen anderer Plattformnutzer teilhaben. Das Repository für komposite Anwendungen enthält neben den Kompositionsfragmenten, die durch die iterative Entwicklung mit den zuvor erwähnten Beispielkomponenten entstehen, auch 26 Anwendungen, die für Benchmarks und zum Testen der Konzepte, beispielsweise für die Berechnung aggregierter Qualitätseigenschaften oder die Empfehlungsprozesse, erstellt wurden. Die Anwendungsmodelle liegen als Mashup Composition Model (MCM) vor und enthalten zusammen 3487 XML-Knoten. Eine einfache Beispielanwendung davon ist als Screenshot in Abbildung 7.3 aus der Domäne Tourismus zu sehen. Sie zeigt eine Kartenkomponente und eine Komponente für die

Suche nach Sehenswürdigkeiten (englisch: points of interest). Sie sind über Kommunikationskanäle derart verknüpft, dass der jeweils angegebene Ort als Input die Aktualisierung der Ansicht der anderen Komponente auslöst.

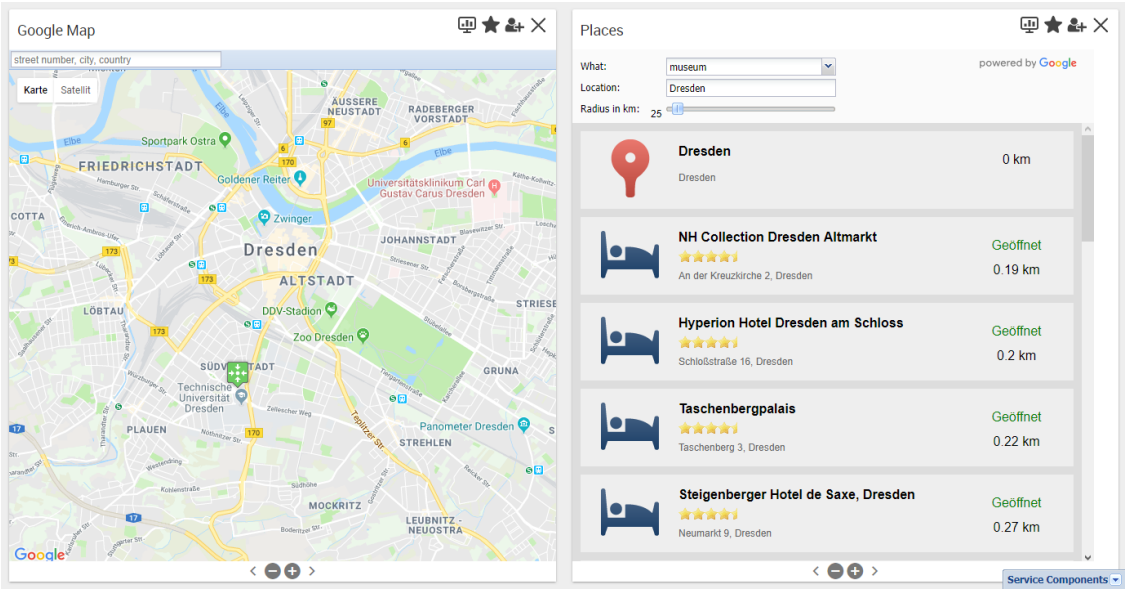

Abbildung 7.3: Komposite Nutzerschnittstelle einer Beispielanwendung aus der Domäne Tourismus in der CRUISE-Plattform

7.4 Implementierung der Referenzarchitektur

Neben den Modellen und Werkzeugen sind die Repositorys für Kompositionsfragmente (CoRe und AppRe), die Mashup-Laufzeitumgebung (MRE) sowie einige zusätzliche Web-Services essenzielle Bestandteile der implementierten Referenzarchitektur in der Mashup-Plattform CRUISE. Dieser Abschnitt beschreibt zunächst die Evaluatoren im CoRe und in der MRE, die für das Auswerten von Qualitätseigenschaften benötigt werden. In diesem Zusammenhang wird auch auf die Umsetzung der Fuzzy-Logik mit den Konzepten der Fuzzy-Mengen als Modelle und der Fuzzy-Terme, die den Qualitätseigenschaften zugeordnet sind und die innerhalb der Anforderungen verwendet werden, eingegangen. Für die Verwaltung von gesammelten Eigenschaftswerten wurde eine Aggregationsinfrastruktur geschaffen, die als Teil des CoRe mit einem REST-basierten Dienst für das Zusammentragen und Verrechnen der Einzelbewertungen verantwortlich ist. Analog dazu wird anschließend die Umsetzung der Kontextsensorik exemplarisch an ausgewählten Beispielen beschrieben. Zusätzlich zu den als Algorithmen und Funktionen dargestellten Teilen der Implementierung in Abbildung 7.1 wurden einige zusätzliche Dienste wie der Kontextdienst und der Nutzerdienst umgesetzt bzw. erweitert, zu denen am Ende des Abschnitts Stellung genommen wird.

7.4.1 Fuzzy-Mengen und Fuzzy-Terme

Die Fuzzy-Auswertung und damit die Nutzung von Fuzzy-Mengen, die den Fuzzy-Termen zugeordnet sind, wird benötigt, wenn Qualitätsanforderungen Fuzzy-Bedingungen enthalten. Diese Unterscheidung wird im Auswertungsprozess beim Prüfen des Bedingungsbaumes getroffen, vgl. Abbildung 5.9. Voraussetzung für die Fuzzy-Auswertung ist die Zuordnung von Fuzzy-Mengen (englisch: fuzzy sets) zu den verwendeten Termen und Typen von Qualitätseigenschaften. Zum Erproben einer praktikablen Menge an Beispieldaten erfolgte zunächst die Zuweisung von Termen für jede Qualitätseigenschaft aus dem Referenzmodell, vgl. auch Abschnitt 4.6. Für die Beispielanforderungen aus den Szenarien wurden auch Vorgabefunktionen als Fuzzy-Mengen unter Zuhilfenahme des Fuzzy-Set-Editors erstellt, die als TURTLE-Modelle vorliegen und der Auswertungsinfrastruktur zu Verfügung stehen. In Abbildung 7.4 ist beispielhaft das Fuzzy-Set für die Qualitätseigenschaft *Response Time* zu sehen. Die Punktkette besteht dabei aus zwei Punkten. In der ersten Zeile sind zudem die anwendbaren Terme *Low*, *Medium* und *High* für die Eigenschaft beschrieben. Neben der Zuordnung des Terms *Low* zur Funktion ist auch zu erkennen, wie die modellierten Daten als Funktionsgraph dargestellt werden können.

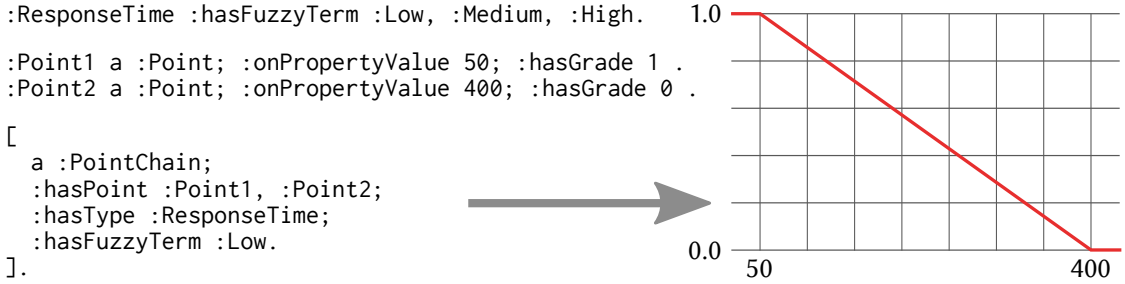

Abbildung 7.4: Serialisierung von Fuzzy-Sets im Datenmodell als TURTLE und der zugehörige Funktionsgraph

7.4.2 Evaluator für Qualitätsanforderungen

Wie in Unterabschnitt 5.5.2 konzipiert, wurde die Auswertung von Qualitätsanforderungen an mehreren Stellen umgesetzt. Für messbare Eigenschaften existiert ein Evaluator in der Laufzeitumgebung und für die anderen beiden Typen der Wertermittlung im Repository. In der MRE wird der Evaluator als Umsetzung in JavaScript direkt vom Qualitätsmonitor angesprochen. Der Evaluator im CoRe hat zusätzlich Zugriff auf die aggregierten Daten der gesammelten Werte für Qualitätseigenschaften. Er wird über den Empfehlungsdienst (englisch: recommendation service) angesprochen, der als REST-basierter Web-Service zur Verfügung steht und von der Laufzeitumgebung genutzt werden kann. Alle Typen der Wertermittlung können dabei auch gemischt in einer einzigen Qualitätsanforderung auftreten, wie es in Abbildung 5.8 zu sehen ist.

Die komposite Struktur der Anforderung wird auf die Struktur der Auswertungsergebnisse projiziert. Dabei werden auch für jeden Zwischenschritt der Komposition Teilergebnisse berechnet. Ausschlaggebend ist insgesamt das Auswertungsergebnis für den ganzen Bedingungsbaum, das mit der Wurzelbedingung assoziiert wird, die direkt mit der Anforderung auf Modellebene verknüpft ist. Die Teilergebnisse werden mitgeliefert, um sie beispielsweise während der

Empfehlung für zusätzliche Awareness-Funktionen zu benutzen. Soll die Qualitätsbewertung Teil einer komplexeren Empfehlung sein, wird sie dem Empfehlungsdienst als Teilergebnis zugeführt, der sie dann entsprechend einer Wichtungskonfiguration mit anderen Bewertungen verrechnet. Ein solcher hybrider Empfehlungsprozess wird in [Rad+12] beschrieben.

7.4.3 Verwaltung von Qualitätseigenschaften

Außer bei den messbaren Qualitätseigenschaften, deren Werte nur transient im System auftreten, müssen die festgelegten und gesammelten Werte über eine Verwaltungsinfrastruktur bereitgestellt werden. Für das Parsen von Wertebelegungen, die für Komponenten innerhalb der SMCDL vom Komponentenautor festgelegt sind, übernimmt der in dieser Arbeit erweiterte *Konverter* (englisch: converter) das Erstellen des OWL-Modells, das innerhalb des CoRe vorgehalten wird. Von dort aus werden die Wertebelegungen ausgelesen und auch zum Zweck der Auswertung von Anforderungen bereitgestellt. Bei Werten für Qualitätseigenschaften des Typs *gesammelt* müssen zusätzlich historische Werte vorgehalten werden. Außerdem wird eine Infrastruktur für das Hinzufügen neuer Einzelwerte und die Aggregation dieser in einen über das Komponentenmodell auslesbaren Einzelwert benötigt. Hierzu wurde im Datenmodell für jede Instanz einer Qualitätseigenschaft eine zusätzliche Speichermöglichkeit geschaffen. Soll beispielsweise das arithmetische Mittel als Aggregationsfunktion genutzt werden, so reicht es nicht aus, wenn der aggregierte Wert der bisherigen Bewertungen vorliegt und eine neue Bewertung damit verrechnet werden soll. Mindestens bedarf es hierbei der Kenntnis der Anzahl der bisherigen Bewertungen oder – als universelle Strategie für das Anwenden anderer Aggregationsvorschriften – der Kenntnis aller bisherigen Bewertungen. Im zweiten, hier präferierten und implementierten, Fall wird der aggregierte Wert direkt aus den Basiswerten berechnet. Ein *Manager für die Verwaltung von Qualitätseigenschaften* nimmt die Aggregation vor und speichert die Ergebniswerte in das Datenmodell, sodass sie über den üblichen Weg wie die Werte anderer Typen von Qualitätseigenschaften ausgelesen werden können. Er ist außerdem für das Entgegennehmen neuer Werte und das Zurücksetzen gesammelter Werte verantwortlich. Er nutzt dafür einen REST-basierten Web-Service. Ein Screenshot der Schnittstellen dieses Dienstes für die Manipulation gesammelter Werte ist in Abbildung E.2 zu sehen.

Er ist ein Bestandteil der Service-basierten Verwaltungsinfrastruktur des CoRe. In Abbildung E.1 ist die in dieser Arbeit neu hinzugekommene REST-Schnittstelle im CoRe als Screenshot des Swagger-UI dargestellt. Neben der Manipulation der gesammelten Qualitätseigenschaften lässt das API auch das Auswerten von Qualitätsanforderungen, das Registrieren und Abrufen von Mashup-Komponenten sowie das Verwalten semantischer Modelle mit SPARQL zu. Als Erweiterung der CRUISE-Infrastruktur bietet es den beteiligten Clients, vor allem der auf JavaScript basierenden Laufzeitumgebung, eine komfortable Zugriffsmöglichkeit. Das in der Abbildung gezeigte Web-Frontend wird über Annotationen und JSR-Standards, vor allem dem Java API for RESTful Web Services (JAX-RS) mit der Referenzimplementierung *Jersey*, erzeugt und bietet hilfreiche Werkzeuge für das Entwickeln und Testen in einer verteilten Umgebung.

Der hybride Auswertungsprozess benötigt eine Schnittstelle für das Bereitstellen von Empfehlungen bzw. Bewertungen (englisch: *ratings*). Der Dienst erfüllt diese Aufgabe mit zwei Methoden. Die Methode `POST ratings` nimmt eine in TURTLE formulierte Qualitätsanforderung sowie einen optionalen Filterausdruck entgegen. Daraufhin werden alle in Frage kommenden Kandidaten gegen die Anforderung geprüft und die Antwort enthält eine Datenstruktur, die

pro Kandidat eine Gesamtbewertung sowie wenige Metadaten enthält. Über die weitere Methode `POST rating` kann eine detaillierte Bewertung mit Teilbewertungen für einen bestimmten Kandidaten angefordert werden, der beispielsweise über einen Komponentenbezeichner festgelegt wird. Sollten in der Laufzeitumgebung bereits Teilbewertungen für die hybride Auswertung entstanden sein, werden diese als weitere Parameter mitgeliefert.

Das Verwalten von Komponenten über ihre SMCDLs erfolgt ebenfalls mit Hilfe eines Web-Service. Über `GET components`, optional mit Filterausdruck als Parameter, wird eine *Liste aller registrierten Komponenten* als JSON-Datenstruktur geliefert. Eine *bestimmte Komponente* kann über ihren Bezeichner als `GET component` angefragt werden. Als Antwort erhält man den Komponentendeskriptor als XML-Struktur. Das *Registrieren* neuer Komponenten und das *Aktualisieren* bereits registrierter Komponenten wird über die Schnittstelle `PUT component` behandelt. Hier sind neben dem XML-Deskriptor als Parameter noch Nutzername und Passwort als Zugriffskontrolle notwendig. Dies gilt auch für das *Löschen* einer Komponente über `DELETE component` mit der Angabe des Komponentenbezeichners.

Eine weitere Schnittstelle bietet einen direkten Zugriff auf semantische Modelle im Repository mit SPARQL bzw. Änderung über die entsprechende Update-Sprache. Dabei wird ein gesondertes Modell genutzt, auf dem die Änderungen ausgeführt werden. Es wird für das Hinzufügen von Termen verwendet, beispielsweise vom Fuzzy-Set-Editor als Client. Auch zusätzliche Daten können so dem Repository zugänglich gemacht werden. `POST update` setzt mit einem SPARQL-Update-Ausdruck neuen Inhalt im Modell. `POST query` erlaubt das Anfragen von Modellinhalten aller eingepflegten Modelle über eine SPARQL-Query-Anfrage.

7.4.4 Kontextsensorik

Die exemplarische Implementierung der Kontextsensorik konzentriert sich auf die über JavaScript verfügbaren Wertebelegungen für Qualitätseigenschaften. Codeausschnitt C.1 zeigt in der `show`-Methode einen einfachen Sensor für die Eigenschaft *ResponseTime*. Darüber hinaus sind auch Plugin- bzw. Hybridtechniken wie *Apache Cordova* sowie weitere APIs der Laufzeitumgebung ansprechbar, die vor allem das Sensorikpotenzial mobiler Geräte ausnutzen können. Diese Szenarien werden detailliert in der Arbeit von Mroß u. a. zu Multi-Device-Mashups mit mobilen Geräten behandelt, siehe [MM14]. Auch aggregierte Kontextdaten, die ggf. externe Verarbeitungsschritte durchlaufen haben, sind Kandidaten für Erweiterungen des Eigenschaftsmodells. Hierfür wird üblicherweise ein Kontextdienst angebunden, der die Daten seiner Kontextmodelle über einen Web-Service bereitstellt. Diese Kontextmodelle koexistieren mit den semantischen Daten zur Profilverwaltung für die Nutzer der Mashup-Plattformen, die unter anderem Informationen über beliebte Qualitätsanforderungen enthalten.

7.4.5 Services in der Mashup-Plattform

Im Component Repository (CoRe) werden neben den Aufgaben der Auslieferung und Empfehlung von Mashup-Komponenten auch Anwendungsmodelle über das enthaltene Application Repository (AppRe) bereitgestellt. Außerdem sorgt ein *Mediationsdienst* in diesem Zusammenhang für das Transformieren von syntaktisch zunächst inkompatiblem Datentypen. Da diese Funktionalität im Rahmen dieser Arbeit jedoch nicht im Fokus steht, wird sie nur informativ erwähnt. Parallel zu den in dieser Arbeit entwickelten REST-Schnittstellen, vgl. oben, können

auch die bisherigen auf XML und SOAP basierenden Schnittstellen *CoreService*, *AppreService* und *RecoService* für die Empfehlung von Kompositionsfragmenten mit eingeschränkter Funktionalität genutzt werden.

Der Kontextdienst ist unter anderem für das Ablegen von Nutzerprofilen mit den für diese Arbeit relevanten Qualitätsanforderungen, die als Favoriten gespeichert werden, verantwortlich. Zur Umsetzung wird *Fuseki*[2], ein Store für RDF-Tripel mit Web-Schnittstelle, genutzt. Neue und Aktualisierungen für existierende Anforderungen werden mittels SPARQL Update übermittelt und mit SPARQL 1.1 für bestimmte Nutzer abgerufen. Die Ergebnisse stehen als Datenstruktur basierend auf JSON in der verteilten Laufzeitumgebung zum Transport bereit. *Jena* kann sie automatisiert aus der Antwort des Kontextdienstes erzeugen. Ein Ausschnitt aus dem Metamodell für das Zuordnen von Qualitätsanforderungen zu Nutzern der Laufzeitumgebung ist in Codeausschnitt 7.1 zu sehen. Der Nutzerbezeichner (UID) wird dabei mit der laufenden Session in der Laufzeitumgebung assoziiert.

Codeausschnitt 7.1: Ausschnitt aus dem Kontextmetamodell um Qualitätsanforderungen für Nutzer als Favorit zu speichern

```
@prefix : <http://mmt.inf.tu-dresden.de/models/quality/person-requirement#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix qreq: <http://mmt.inf.tu-dresden.de/models/quality/requirement#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://mmt.inf.tu-dresden.de/models/quality/person-requirement>
  rdf:type owl:Ontology ;
  dc:description "Quality properties metamodel for having person requirements for components"@en ;
  dc:title "Web Mashup Person Requirement Properties"@en ;
  owl:imports <http://mmt.inf.tu-dresden.de/models/quality/property> ;
  owl:imports <http://mmt.inf.tu-dresden.de/models/quality/requirement> ;
  owl:imports <http://xmlns.com/foaf/0.1/> .

:Person
  rdf:type owl:Class ;
  rdfs:comment "Class for user specific quality requirements"@en ;
  rdfs:subClassOf <http://mmt.inf.tu-dresden.de/models/quality/property#QualityPropertyCarryingEntity> ;
  rdfs:subClassOf <http://xmlns.com/foaf/0.1/Person> .

:hasUID
  rdf:type owl:DatatypeProperty ;
  rdfs:domain :Person .
```

Mashup-Komponenten referenzieren über ihre Deskriptoren eine Vielzahl verteilter Ressourcen. Für die originären Ressourcen von Mashup-Komponenten wird ein Repository bereitgestellt, das die Auslieferung übernimmt. Drittressourcen, wie UI-Frameworks werden möglichst über Content Delivery Networks (CDNs) ausgeliefert, um sowohl eine hohe Verfügbarkeit als auch das DRY-Prinzip durch das Vermeiden von Dopplungen für JavaScript-, CSS- und Bildressourcen umzusetzen. Netzwerke, die beispielsweise genutzt werden, sind *CDNJS* und *StackPath*.

[2] Apache Jena Fuseki: jena.apache.org/documentation/fuseki2/

7.5 Implementierung der Werkzeuge

Die bereits im Überblick vorgestellten Werkzeuge haben in der Mashup-Infrastruktur dieser Arbeit zwei Hauptziele. Zum einen bieten sie dem Nutzer der Laufzeitumgebung und dem Ausführenden administrativer Aufgaben elementaren Zugang zu den von den Web-Services angebotenen Funktionen, zum anderen ermöglichen sie eine integrierte Entwicklung und Nutzung nach dem Prinzip der iterativen Live-Sophistication. Zunächst werden Details der Umsetzung im *Anforderungsassistenten*, der konzeptionell im Rahmen der Spezifikation von Qualitätsanforderungen eingeführt wurde, beschrieben. Der *Editor für Fuzzy-Sets* ermöglicht das Zuordnen, Festlegen und visuelle Ändern von Fuzzy-Sets als administrative Aufgabe. Im *Anwendungsbrowser* kann der Nutzer Hilfestellung bei der Auswahl von Mashups erhalten, indem er sich über aggregierte Qualitätseigenschaftswerte informiert. Schließlich wird der *Komponentenbrowser* vorgestellt, der ein Hilfsmittel für das Verwalten von Mashup-Komponenten ist.

7.5.1 Anforderungsassistent

Wichtigstes Werkzeug zum Erstellen bzw. Laden von Qualitätsanforderungen in der Laufzeitumgebung ist der Anforderungsassistent (englisch: requirements wizard), der in zwei Modi in Erscheinung tritt, vgl. Abschnitt 5.4. Im *Empfehlungsmodus* wird er ganz zu Beginn der Plattformnutzung im Dashboard nach dem Login gezeigt. Alternativ kann er zur Unterstützung bei jeder Entwicklungsaktion aktiviert werden, beispielsweise beim Hinzufügen von Kompositionsfragmenten. Abbildung 7.5 zeigt den Anforderungsassistenten als Screenshot im Empfehlungsmodus. Der visuelle Aufbau der beiden Modi orientiert sich an der konzeptionellen Skizze, siehe Abbildung 5.4. Die Editoren für die Auswahl der *Goals* und für das Komponieren von *Conditions* teilen sich ein *Akkordeon* als UI-Element. In der Abbildung ist der Goal-Editor zugeklappt, da er für diese Arbeit nicht im Fokus steht. Die Überschrift »What do you want to do?« ermuntert zur Eingabe von Capabilitys, die die Goals beschreiben, wie beispielsweise *show weather* oder *find route* mit der typischen Paarung aus Aktivität und Entität. Diese Capabilitys werden aus einer Menge von Vorschlägen ausgewählt die das Empfehlungssystem aufgrund der Nutzerhistorie, der aktuellen Anwendungskonfiguration und weiteren Kriterien auswählt.

Zweck des Condition-Editors ist das Verfeinern der Anforderungen während der Empfehlung. Durch das Setzen von Bedingungen als Qualitätsanforderung kann somit die Aufgabe »Improve results« wahrgenommen werden. Je nach Nutzerfähigkeiten wird ein Qualitätsprofil mit hinterlegtem Bedingungsbaum ausgewählt oder die Bedingung mit den Einzelangaben manuell erstellt und bearbeitet. Einen weiteren Screenshot des Assistenten mit dem Editor für Fuzzy-Bedingungen mit Termauswahl und der Markierung als Favorit zeigt Abbildung D.2.

Der Condition-Editor ist wie folgt aufgebaut. Zunächst kann binär entschieden werden, ob man die Bedingung mit Hilfe eines Vergleichswertes und Operators oder mit der Auswahl eines oder mehrerer Fuzzy-Terme aufbauen möchte. Danach wird die gewünschte Eigenschaft ausgewählt. Hierbei ist auch das hierarchische Verketten möglich, indem mit + eine weitere Ebene in der Eigenschaftskette betreten wird. Im Falle der diskreten Bedingung kann nun der Operator als Drop-Down-Eintrag und der Vergleichswert in einem Textfeld festgelegt werden. Soll eine Fuzzy-Bedingung entstehen, muss der Nutzer aus den angezeigten Termen diejenigen auswählen, die gelten sollen. In Abbildung 7.5 sieht man auch als Hilfestellung die Einheit, für die ein Vergleichswert einer diskreten Bedingung eingetragen wird. Jeder Bedingungsbaum erscheint

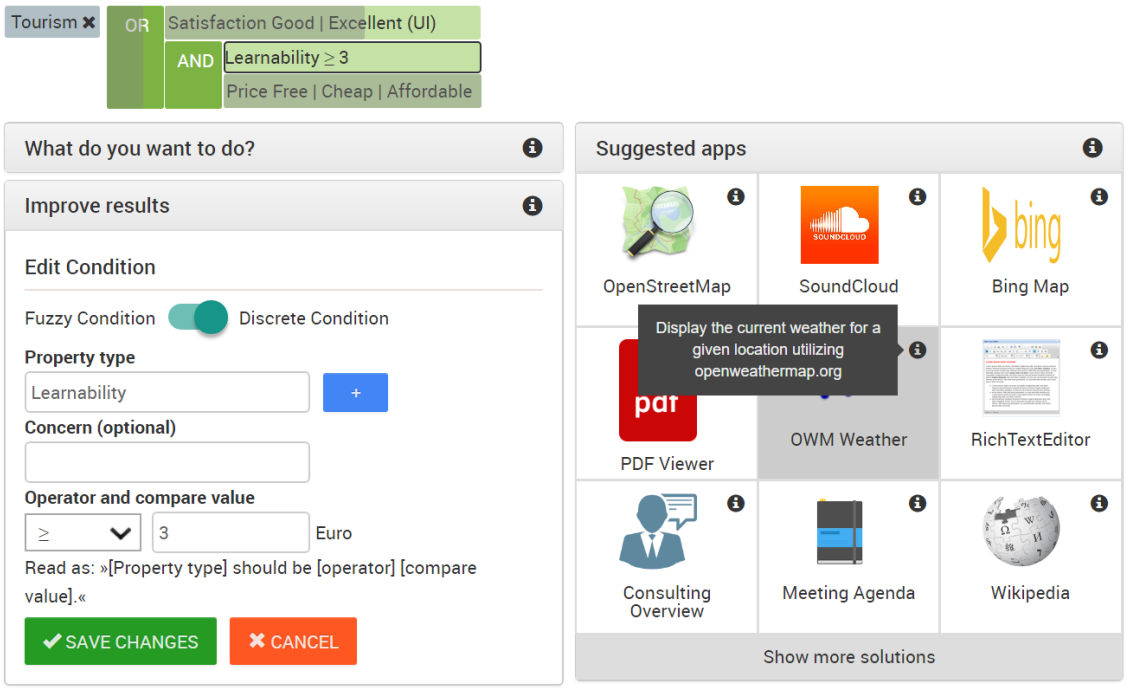

Abbildung 7.5: Anforderungsassistent im Empfehlungsmodus zum Spezifizieren von Fähigkeiten und Qualitätsanforderungen für neue Kompositionsfragmente

oben in der Anforderungsleiste als Box zur Navigation. Dort ist über Hover-Schaltflächen auch das Markieren einzelner Bedingungen für das Bearbeiten möglich. Die Veränderungen werden dann wieder im Condition-Editor für die aktuell geltende Auswahl vorgenommen.

Zu den so definierten Anforderungen passende Kompositionsfragmente werden als Vorschläge angezeigt. Da viele Nutzer Anwendungen oder Teile davon, die eine bestimmte Aufgabe erfüllen, von den Plattformen ihrer Mobilgeräte als *Apps* kennen, ist dieser Bereich mit »Suggested apps« überschrieben. Bei jeder Veränderung der Bedingungen wird die Anforderungsauswertung angestoßen und die Liste der vorgeschlagenen Kompositionsfragmente aktualisiert. Die Sortierung nach dem Auswertungsergebnis sorgt dafür, dass die passendsten Kandidaten ganz oben stehen. Beim Hovern über einen Vorschlag wird oben in der Anforderungsleiste in jeder Bedingung ein Indikator als Streifen eingeblendet, der den Erfüllungsgrad anzeigt. Die Indikatoren innerhalb der Boxen für die Junktoren zeigen den Erfüllungsgrad für den jeweiligen Teilbaum an. Die Anforderungsleiste setzt das Laden und Speichern favorisierter Anforderungen aus dem Kontextmodell des angemeldeten Nutzers um. Zum Aktivieren und Deaktivieren von Favoriten wird der entsprechende Button in Icon-Form am Bedingungsbaum betätigt, vgl. *Herz-Symbol* in Abbildung D.2.

Der Anforderungsassistent kommt im Überwachungsmodus ohne das Konfigurieren und Anzeigen von Domänen und Capabilitys im Speziellen aus, ergänzt jedoch den Editor für Actions und Events. Abbildung 7.6 zeigt, dass oben in der Anforderungsleiste mehrere Aktionen – und damit auch Events – einem Bedingungsbaum zugeordnet werden können. Grundsätzlich kann in den beiden Editoren eine Auswahl aus dem jeweiligen Referenzmodell getroffen werden. Da-

bei stehen spezifische Interaktionselemente, wie beispielsweise ein Slider für Zeitintervalle oder ein Textfeld für das Eingeben von Log-Nachrichten bzw. eine Auswahl von Kompositionsfragmenten für das automatisierte Ersetzen, bereit.

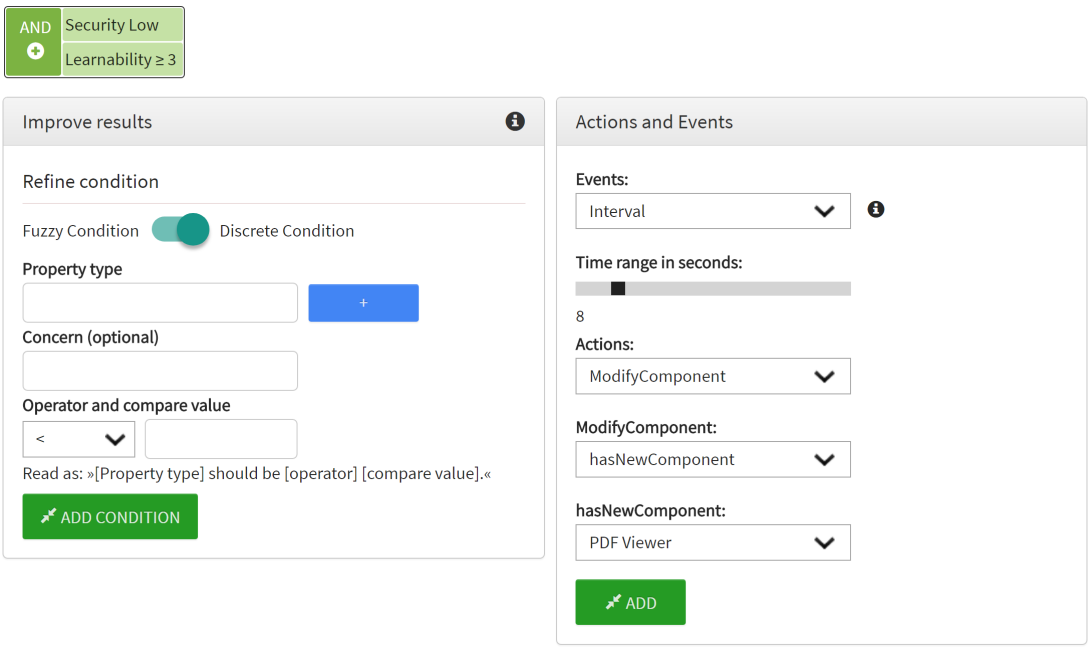

Abbildung 7.6: Anforderungsassistent im Überwachungsmodus zum Spezifizieren von Laufzeitanforderungen mit Überwachungsereignissen und Adaptionsaktionen

Die Laufzeitanforderungen können für verschiedene Scopes definiert werden. Dazu wird der Assistent entweder über das Plattform-Menü oder den Indikator einer Komponente geöffnet. Abbildung D.3 zeigt diese beiden Möglichkeiten. Die Titelleiste des Assistenten zeigt jeweils an, in welchem Scope man sich befindet, sobald der Assistent geöffnet ist. Bei Komponenten wird der Name der Komponente als Hinweis angezeigt.

7.5.2 Editor für Fuzzy-Mengen

Für das Spezifizieren von Fuzzy-Bedingungen – beispielsweise im Anforderungsassistent, vgl. Unterabschnitt 7.5.1 – ist das Vorhandensein entsprechender Terme erforderlich. Diese müssen mit Zugehörigkeitsfunktionen hinterlegt sein, um eine sinnvolle Anforderungsauswertung zu gewährleisten. Bei der Auswertung werden diese Funktionen aus einem semantischen Modell gelesen. Da das initiale Erstellen und Anpassen auf der Ebene von RDF bzw. OWL mühselig ist, steht mit dem *Fuzzy-Set-Editor* ein Werkzeug zum Erzeugen, Bearbeiten, Suchen und Verwalten von Zugehörigkeitsfunktionen bereit, das über Web-Services an die Auswertungsinfrastruktur gekoppelt ist.

Zunächst können in der ersten Ansicht – vgl. Abbildung 7.7 – Funktionen über Eigenschaftstypen und ihre zugeordneten Terme gesucht werden. Dies geschieht über die beiden Auswahlfelder oben. Zu einem Eigenschaftstyp werden direkt darunter ein kurzer Beschreibungstext, die

im Editor genutzten Begrenzungen für die Eigenschaftswerte (englisch: *domain limits*) und sinnvolle Terme angezeigt. Darunter sieht man die im System gefundenen Fuzzy-Sets. Mit dem Klick auf ein bestimmtes Fuzzy-Set – hier: *Satisfaction Good* erscheint rechts daneben eine Vorschau auf die verknüpfte Zugehörigkeitsfunktion. Über die in der jeweiligen Zeile vorhandenen Icons lässt sich der Bearbeitungsmodus für die ausgewählte Funktion erreichen bzw. die Funktion löschen. Sollte es für einen Eigenschaftstyp und einen gewünschten Term noch kein Fuzzy-Set geben, so lässt es sich an dieser Stelle erzeugen. Gleiches gilt für noch nicht vorhandene Terme.

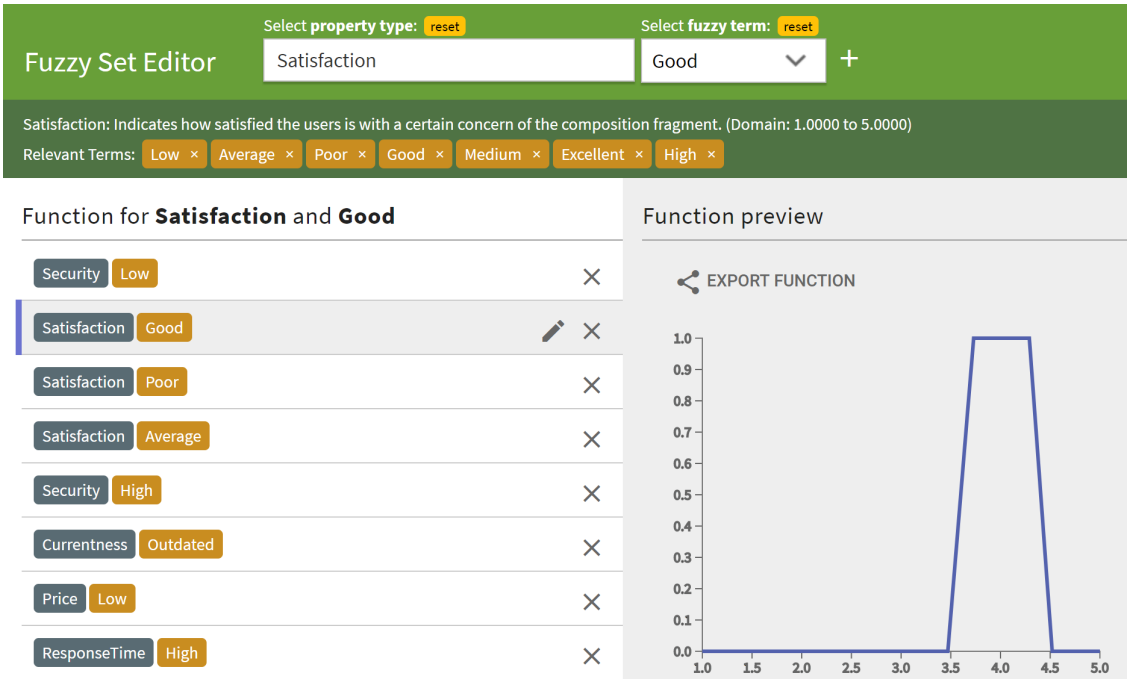

Abbildung 7.7: Fuzzy-Set-Editor: Filtern und Auswählen von Zugehörigkeitsfunktionen auf Basis von Eigenschaftstypen und Termen

Der Bearbeitungsmodus in Abbildung 7.8 zeigt den Funktionsgraphen des Fuzzy-Sets, das zuvor ausgewählt wurde, in einer großen Ansicht. Zunächst kann der Viewport mit den Operationen zum Verschieben und Strecken, die auch über Tastennavigation verfügbar sind, zu interessanten Stellen bewegt werden. Auf der Oberfläche sind die Modifikatoren dieser Begrenzungen des Definitionsbereiches unten links als Icon-Schaltflächen mit Pfeilen und Lupen gekennzeichnet. Die Limits gelten jeweils für alle Funktionen eines Eigenschaftstyps. Neben der Veränderung des Typs und des Terms ist über die Schaltflächen oben rechts das Speichern, das Löschen aller Eckpunkte, das Anzeigen einer Detailbeschreibung zum Typ und das Verlassen des Bearbeitungsmodus möglich. Der Funktionsgraph selbst kann über die anfassbaren Eckpunkte verändert werden. Beim Verschieben der Anfasser mit der Maus zeigt ein Tooltip jeweils die aktuellen Koordinaten an. Der Funktionswert als Zugehörigkeitsgrad eines Eigenschaftswertes im Fuzzy-Set liegt dabei immer zwischen 0 und 1, vgl. Unterabschnitt 4.4.1. Doppelklicks auf die Anfasser entfernen den Eckpunkt und die Funktion verbindet die bis dahin benachbarten Eckpunkte linear. Neue Eckpunkte werden über einen Klick auf die Verbindungslinien hinzugefügt. Die äußersten Eckpunkte verlängern die Funktion unter gleichbleibendem Funktionswert

seitlich ins Unendliche. Als Overlay sieht man in Abbildung 7.8 außerdem eine Übersicht aller Funktionsgraphen einer Qualitätseigenschaft. Diese Übersicht wird auf Wunsch vom Editor generiert. Sie erlaubt das Beurteilen von Überschneidungen und gibt einen Gesamteindruck aller Terme und ihrer Funktionen für eine bestimmte Eigenschaft. Zum persistenten Speichern der Funktionen und Zuordnungen im Editor wird ein SPARQL-Endpoint angebunden. Sowohl der oben vorgestellte semantische Kontextdienst als auch der Verwaltungsdienst des CoRe wurden hierfür praktisch erprobt.

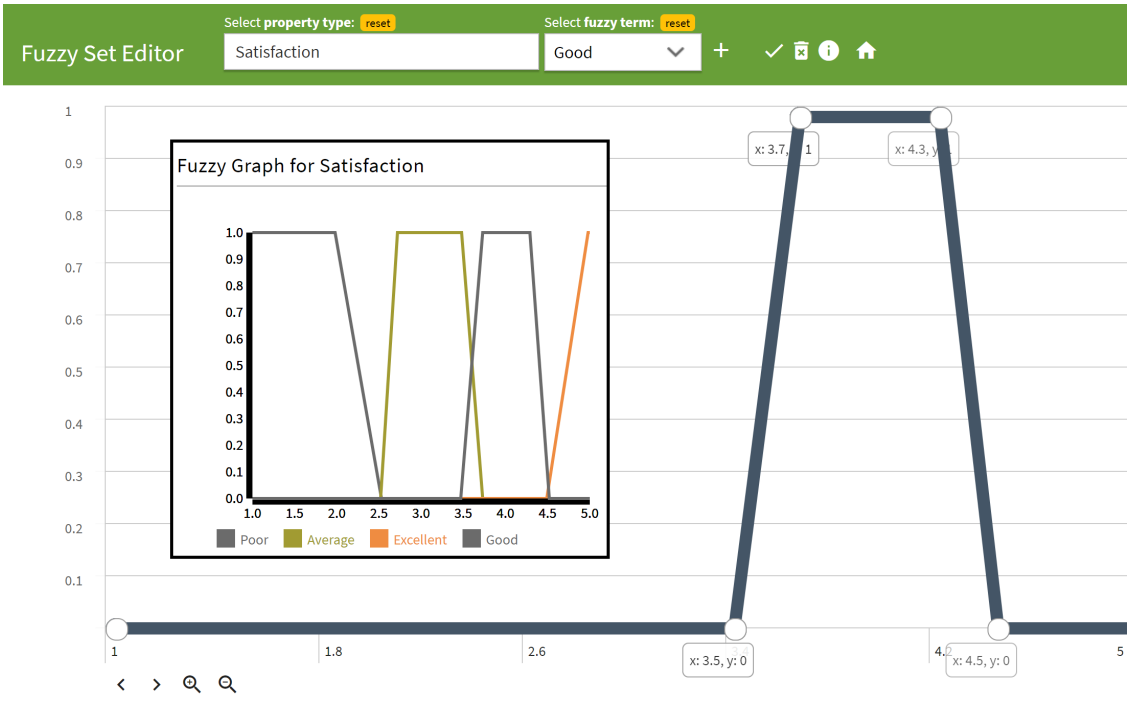

Abbildung 7.8: Fuzzy-Set-Editor: Graphisches Bearbeiten einer Zugehörigkeitsfunktion

7.5.3 Anwendungsbrowser mit Aggregationssicht und Komponentenbrowser

Ein weiteres Werkzeug, das sich sowohl an Anwendungsnutzer als auch an administratives Personal richtet, ist der Anwendungsbrowser. Er ist in erster Linie ein Frontend für das Application Repository (AppRe) und bietet somit den Zugang zum Anwendungsmarktplatz mit der direkten Möglichkeit des Startens einer Anwendung in der Laufzeitumgebung. Abbildung 7.9 zeigt die Einträge für drei Anwendungen als Inhalte eines Beispielrepositorys. Die Metadaten, die für registrierte Mashups im Anwendungsbrowser angezeigt werden, konzentrieren sich hauptsächlich auf die Wertebelegungen der Qualitätseigenschaften aus dem Referenzmodell. So zeigt Abbildung 7.10 beispielhaft die Wertebelegung für eine Mashup-Komponente, die eine Geo-Karte auf Basis von *OpenStreetMap* anbietet. Sie wurde über eine verzeichnete Mashup-Anwendung ausgewählt und bezieht die Wertebelegung für die Qualitätseigenschaften der Typen *Provided* und *Collected* aus den Schnittstellen des entsprechenden CoRe.

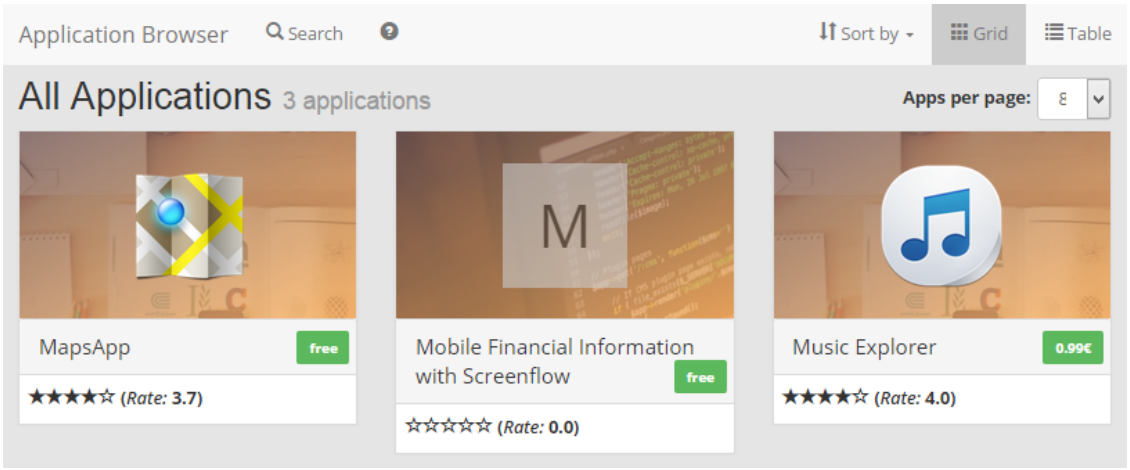

Abbildung 7.9: Anwendungsbrowser: Anwendungsübersicht

Der wichtigste Mehrwert dieses Browsers ist das *Aggregieren* von Eigenschaftswerten einzelner Mashup-Komponenten einer Anwendung zu jeweils einem singulären Wert, der die Eigenschaft für das gesamte Mashup und eine bestimmte Eigenschaft repräsentiert. Neben der typbasierten Aggregationsvorschrift, die im Anwendungsbrowser nutzergetrieben modifiziert werden kann, ist auch die Visualisierung spezifisch für den *Datentyp* der Qualitätseigenschaft. In Abbildung D.1 ist eine aggregierte Ansicht der Wertebelegungen einer bestimmten Mashup-Anwendung zu sehen. Die Visualisierung orientiert sich dabei an den Vorgabeeinstellungen für den jeweiligen Typ der Qualitätseigenschaft. Dabei ist angegeben, welche Aggregationsfunktion verwendet wird und aus wie vielen Einzelwerten sich der aggregierte Wert zusammensetzt.

Ein alternativer Browser für Mashup-Komponenten, die in einem bestimmten CoRe registriert sind, wurde bereits in Abbildung 7.2 gezeigt. Dieses Frontend dient administrativen Aufgaben und ermöglicht zudem das spezifische Angeben von Abfragefiltern, die jedoch Kenntnisse über SPARQL-Filterausdrücke und die Modellierung der Kompositionsfragmente und der Qualitätseigenschaften erfordern.

7.6 Nutzerstudie zum Anforderungsassistenten

Ein umfassendes Ziel dieser Arbeit ist das erleichterte und effizientere Komponieren situativer Anwendungen mit der Mashup-Technologie als Plattform unter der Nutzung von Qualitätsanforderungen. Diesem Ziel, das sich in Szenario ① einordnet, wird mit dem *Empfehlungsmodus des Anforderungsassistenten* begegnet. Zweck der Untersuchung war es, die Werkzeugunterstützung für die Konzepte dieser Arbeit bei zentralen Aufgaben der Anforderungserstellung und -auswertung auf Nutzertauglichkeit zu überprüfen. Dabei galt es vor allem, die angebotenen Interaktionselemente korrekt zuzuordnen, die präsentierten Ergebnisse im iterativen Prozess der Aufgabenabarbeitung passend zu interpretieren und die Nutzerzufriedenheit einzuschätzen, um die Eignung innerhalb der integrierten Entwicklung zu belegen, vgl. Thesen 2, 3 und 5. Insbesondere bei der Nutzung von Fuzzy-Termen und umgangssprachlichen Anforderungsbedingungen konnte die Eignung der fachlichen Konzepte für die Zielgruppe gezeigt werden. Praktischer

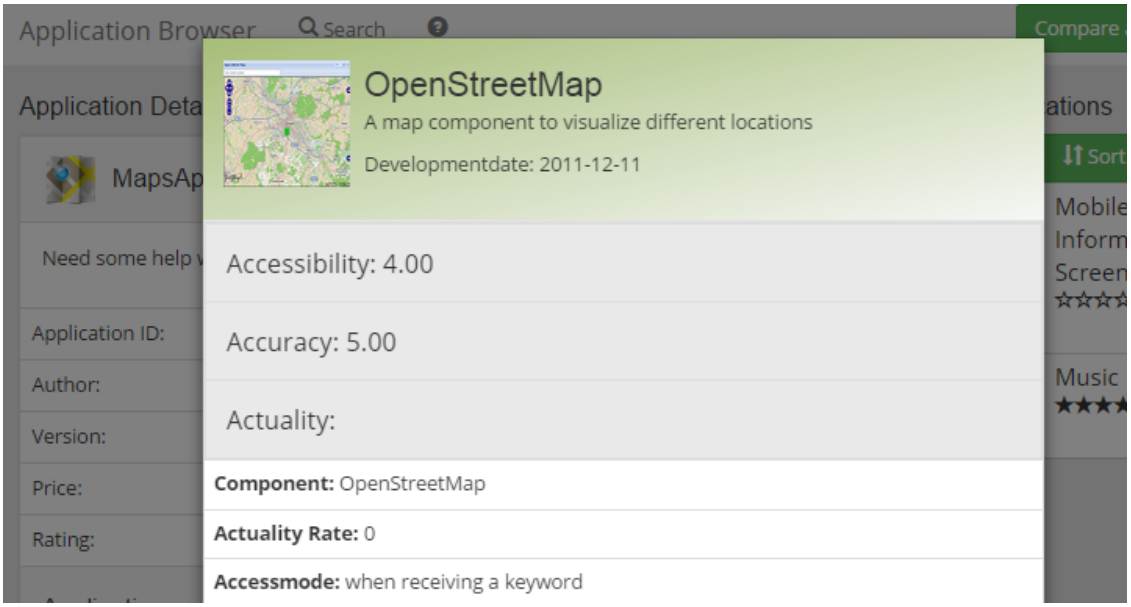

Abbildung 7.10: Anwendungsbrowser: Wertebelegungen der Qualitätseigenschaften für eine zuvor ausgewählte Mashup-Komponente

Untersuchungsgegenstand ist dabei eine Version des Anforderungsassistenten, der in der *TSR* (Thin-Server Runtime Environment) als Vorläufer der CRUISE-Plattform umgesetzt wurde. Diese rein clientseitige Variante der Mashup Runtime Environment (MRE) wird grundsätzlich bereits in [Pie12, Unterabschnitt 7.2.2] beschrieben. Im Rahmen dieser Arbeit wurde diese zu einer verteilten Laufzeitumgebung weiterentwickelt, die auch die aktuelle Ausbaustufe des Anforderungsassistenten enthält. Dabei wurden die Ergebnisse dieser Nutzerstudie eingearbeitet, um das Nutzererlebnis weiter zu verbessern. Diese *Client-Server Runtime Environment (CSR)* erlaubt das Ausführen auf mehreren Geräten in verteilten Anwendungssitzungen und somit auch neue Formen der kollaborativen Nutzung. Abbildung 7.11 zeigt einen Screenshot der TSR-Version des Anforderungsassistenten. Unter ① werden die Kompositionsfragmente vorgeschlagen, die Mitte ② dient zur Angabe von Anforderungen und der Auswahl von Qualitätseigenschaften. Rechts ③ werden die einzelnen Qualitätsanforderungen mit Awareness-Indikatoren für die Erfüllungsgrade dargestellt.

7.6.1 Szenarien und Methodik

Die Teilnehmer, $n = 10$, Lebensalter zwischen 22 und 64 Jahren, mussten verschiedene Aufgaben nacheinander durchführen, die aus der Live-Sophistication in Szenario ① stammen. Mit der Teilnehmeranzahl wurde der Empfehlung von [Fau03] gefolgt, um typischerweise 95 % der Usability-Probleme zu identifizieren. Alle Probanden hatten grundlegende Erfahrungen im Umgang mit Web-Anwendungen, jedoch keine umfassenden Kenntnisse in der Modellierung oder Programmierung von Web-Mashups. Zu Beginn wurde jedem Teilnehmer ein Einführungsvideo[3] in die Mashup-Plattform gezeigt. Dort lernen sie die grundlegenden Möglichkeiten der

[3]Einführungsvideo in die Mashup-Plattform CRUISE mit EDYRA: youtube.com/watch?v=fvuYjpLXrME

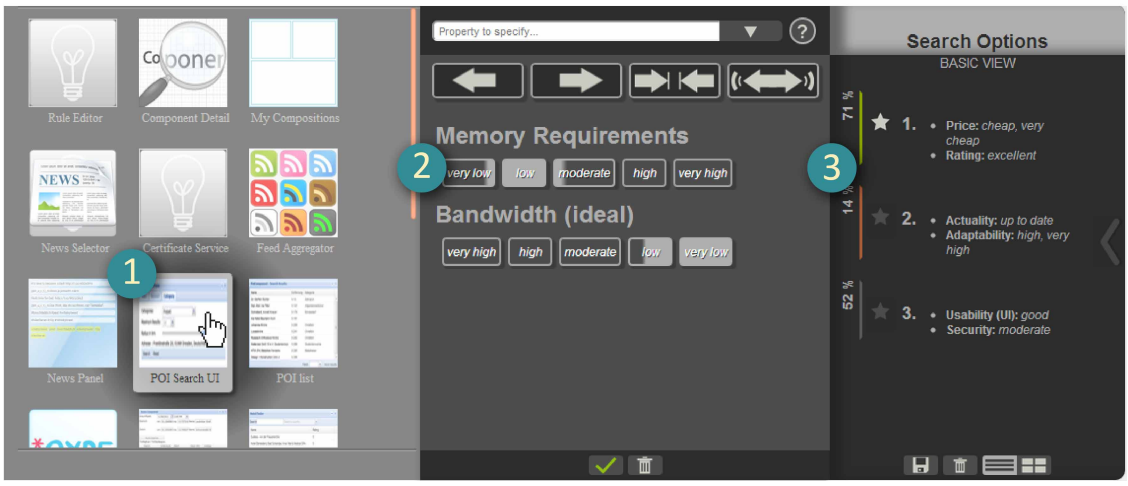

Abbildung 7.11: Anforderungsassistent in der Thin-Server-Laufzeitumgebung

Komposition mit Web-Komponenten, jedoch nicht den Anforderungsassistenten, kennen. Nach der Einführung wurden den Teilnehmern die durchzuführenden Aufgaben vorgestellt, die vor allem das Einbeziehen von Qualitätsanforderungen in den Auswahlprozess für Kompositionsfragmente zum Gegenstand hatten. Zuerst musste die *Suche nach einer spezifischen Mashup-Komponente mit einem Text-Eingabefeld* durchgeführt werden. Dann sollten zusätzliche Qualitätsanforderungen mit Qualitätseigenschaften aus dem Szenario ① einbezogen werden. Dabei galt es zunächst herauszufinden, wie man solche Anforderungen mit Hilfe des Assistenten definiert. Das Ziel hierbei war das *Identifizieren des besten Kandidaten.* Dabei wurde durch Vervielfältigung der Komponenten und leichter Modifikation der Funktionalitäten bzw. Annotationen der Qualitätseigenschaftswerte ein »Rauschen« unter den angezeigten Kandidaten erzeugt, um das Auffinden zu erschweren. Die restlichen Aufgaben beschäftigen sich mit der *Komposition und Konfiguration der Bedingungen*, die innerhalb der Qualitätsanforderungen gelten sollen, und ihrer Repräsentation im Werkzeug.

Während der Durchführung der Aufgaben wurden die Nutzer bei ihrer Bearbeitung der beobachtet. Zusätzlich kam die Methode *Think Aloud* zum Einsatz, in der die Probanden ihre Erwartungen und durchgeführten Einzelaktionen kommentierten. Besonders interessant war dabei der Vergleich der erwarteten Reaktionen des Systems mit den tatsächlich eingetretenen. Dadurch kann validiert werden, ob die Probanden aus der Zielgruppe den intendierten Zweck einer im Werkzeug angebotenen Funktion richtig erkennen bzw. das Bedürfnis haben, diese Funktion zu nutzen bzw. diese in der Plattform zu finden. Nach der Durchführung der Aufgaben wurden die Teilnehmer gebeten, einen Fragebogen über die Interaktionselemente und die Werkzeugunterstützung zur Anforderungskomposition auszufüllen. Sie haben zudem jeweils den standardisierten Fragebogen zur Gebrauchstauglichkeit System Usability Scale (SUS) sowie zur Ermittlung des NASA Task Load Index (NASA-TLX) zur Einschätzung des *Workloads* – d. h. der Belastung in verschiedenen Dimensionen, vgl. unten – ausgefüllt.

7.6.2 Ergebnisse und Konsequenzen für den Assistenten

Der Assistent hatte während der Studie zwei verschiedene Indikatoren für die Awareness der Erfüllung von Qualitätsanforderungen. Alle Teilnehmer kamen mit den senkrechten Streifenindikatoren und deren Prozentanzeige unter ③ gut zurecht. Kaum jemand konnte sich jedoch an weitere Indikatoren – die direkt unter den Kandidaten links auftauchen als aggregierte Bewertungen mit der Berücksichtigung weiterer Kriterien (in der Abbildung nicht dargestellt) – erinnern. Das ging so weit, dass sie nicht einmal sagen konnten, ob die Streifen vor der ersten Anforderungsauswertung bereits vorhanden waren. Durch die nachteilige räumliche Aufteilung wurde das Konzept der Indikatoren für den aktuellen Anforderungsassistenten grundsätzlich überarbeitet, sodass das Feedback für die Erfüllungsgrade an einer Stelle zu sehen ist. Der Speichern-Button zum Übernehmen der Bedingungen aus der Mitte in die rechte Ansicht wurde als sinnvoll und nachvollziehbar angenommen. Vor allem die jüngeren Teilnehmer hätten sich auch vorstellen können, dass die Bedingungen ohne explizite Bestätigung übernommen werden. Auch diese Forderung wurde in die Gestaltung der überarbeiteten Kompositionsansicht übernommen, sodass das Auffinden einzelner Bedingungen in der Hierarchie leichter fällt.

Die zweite Aufgabe bestand im Bearbeiten bestehender Bedingungen. Dabei konnten die Teilnehmer gut mit den Interaktionselementen zum Auswählen einzelner Bedingungen und der Löschfunktion umgehen. Das Unterscheiden der Verknüpfung mit *AND* und *OR* wurde mit Hilfe des Zeitpunktes der Bestätigung in die Übersicht vorgenommen. Überwiegend gut gelang den Nutzern zwar das Interpretieren komplexer Bedingungen, sie hatten jedoch große Probleme mit dem UI-Paradigma der Zuordnung. Deshalb sind die Junktoren nun explizit im Bedingungsbaum zu sehen und können direkt gewechselt werden. Fuzzy-Terme nach Maßgabe der Aufgabenstellung auszuwählen stellte für die Teilnehmer kein Problem dar. Das Konfigurieren der Term-Semantik, vgl. Abbildung 4.7, war jedoch für die meisten Probanden problematisch. Neben der Verfeinerung durch Strecken bzw. Verschieben der Zugehörigkeitsfunktionen wurden auch die Anordnung und die Zuordnung zu möglichen Termen und der entsprechenden Bedingung als verbesserungswürdig eingeschätzt. Da durch diese Art der Konfigurierbarkeit die Ergebnisse nur wenig beeinflusst werden, ist das Strecken und Verschieben der Funktionen nicht mehr Teil des modernen Anforderungsassistenten, um die visuelle und funktionale Komplexität zu reduzieren. Ebenfalls deaktiviert wurde die im Konzept vorgesehene Wichtung zwischen benachbarten Bedingungen. Der untersuchte Assistent sah dafür eine vereinfachte Einstellmöglichkeit mit einem *an- und abwählbaren Stern*, vgl. rechts in Abbildung 7.11, vor. Trotz der großen Nutzerakzeptanz ist sie im aktuellen Anforderungsassistenten nicht mehr zu sehen, da die Interaktionselemente an den Bedingungsboxen bereits eine hohe Interaktionskomplexität erzeugen und eine detailliertere Einstellung der Gewichte nicht zumutbar wäre. In anderen Anwendungsfällen, in denen Anforderungen mit Bedingungsbäumen als Modell importiert oder durch Experten konfiguriert werden, ist die volle Mächtigkeit der Gewichte – vgl. Unterabschnitt 5.3.2 – nutzbar und wird im Auswertungsprozess berücksichtigt. Nutzer, die nicht alltäglich mit Overlays und Popups arbeiten, haben zudem die aufklappenden Teile des Assistenten kritisiert. Diese Aufteilung wurde durch das in Abschnitt 5.4 beschriebene Layout abgelöst, in der das explizite Aufklappen durch die immer sichtbare Anforderungsleiste nicht mehr notwendig ist. Keine Präferenz bzw. Einigkeit gab es bei den Teilnehmern, ob eine vorhergehende Anforderung verschwinden soll, sobald ein neues Kompositionsfragment gesucht wird, beispielsweise über die Text-Eingabe, die Übereinstimmungen nach Capabilitys vornimmt.

Mentale, physische sowie zeitliche Anstrengungen, die eigene Leistung und das Frustrationsniveau konnten im NASA-TLX[4] bewertet werden. Die Teilnehmer vergaben einen Wert von 33 als Median in einer Skala von 0 bis 100. Dies zeigt eine eher niedrige Belastung mit der gestellten Aufgabe. Dieser niedrige Wert ist eine Voraussetzung für die Integration des Assistenten in die Mashup-Plattform, wo er nur ein Teilgebiet der Funktionalität abdeckt. Der SUS-Fragebogen[5] erzeugte Antworten mit einem Median-Wert von 70, welcher eine überdurchschnittliche Usability für das System anzeigt. Weitere Schlussfolgerungen dieses Nutzertests, die Auswirkungen auf die Einordnung in den Entwicklungsprozess und die Laufzeitumgebung innerhalb der Mashup-Plattform haben, werden in Abschnitt 7.8 und Abschnitt 8.2 diskutiert.

7.7 Validierungsergebnisse zu Performance und Awareness-Indikatoren

Durch die Nutzerbeteiligung im Entwicklungsprozess ist eine Einschätzung der Performance des Auswertungsprozesses notwendig, damit die Nutzererfahrung der integrierten Live-Sophistication – vgl. These 5 – nicht gefährdet wird. Zur Sicherstellung wurden einige Performance-Messungen durchgeführt. Dabei stand die *Ausführungsdauer*, d. h. die Laufzeit, für bestimmte Aufgaben im Vordergrund. Die Speicherkomplexität der grundlegenden Datenhaltung und der Ausführung komplexer Berechnungen wurde aufgrund der geringen Größe der verwendeten Artefakte und Beispieldaten nicht gesondert betrachtet.

Im *Anwendungsbrowser*, vgl. Unterabschnitt 7.5.3, der aggregierte Werte für Qualitätseigenschaften auf Anwendungsebene berechnet, wurden vier Testanwendungen mit zwei bis acht Mashup-Komponenten gemessen. Dabei wurde eine Teilmenge des Referenzmodells für Qualitätseigenschaften mit 21 Eigenschaften und somit insgesamt 42 bis 168 Werten genutzt. Die Aggregationsvorschriften, die den Berechnungsalgorithmus bestimmen, waren dabei gemischt. Für den Web-Browser als Ausführungsplattform wurden separate Messungen in Firefox und Chrome durchgeführt. Dabei schnitt Chrome mit einer maximalen Laufzeit von 18 ms etwas schlechter als Firefox mit maximal 12 ms ab. Diese Größenordnung zeigt, dass hier zunächst kein weiterer Optimierungsbedarf besteht, da die Nutzererfahrung durch eine so niedrige Verzögerung praktisch nicht beeinträchtigt ist.

Beteiligt an potenziellen Verzögerungen für den Plattformnutzer ist zudem der Empfehlungsprozess für Kompositionsfragmente, die unter der Nutzung von Qualitätsanforderungen zur Verwendung in einer Mashup-Anwendung vorgeschlagen werden sollen. Die dafür erforderlichen Berechnungen werden verteilt – je nach Art der enthaltenen Bedingungen – ausgeführt, vgl. Unterabschnitt 5.5.2. Die Vergleichsprozesse skalieren dabei in der Hauptsache mit der Anzahl der in der Anforderung enthaltenen Bedingungen und der Anzahl der Kandidaten, mit denen die Wertebelegung verglichen werden muss. Obwohl der Auswertungsprozess immer zeitnah nach jeder Veränderung der Bedingung im Anforderungsassistent vorgenommen wird, ist hierbei keine beeinträchtigende Verzögerung in der Aktualisierung der Kandidaten zu erwarten. Zur Laufzeit des Auswertungsprozesses selbst kommt allerdings noch die Zeit zum Abrufen der Metadaten hinzu, mit denen die Kompositionsfragmente als Vorschläge visualisiert werden – beispielsweise ein Icon, der Titel und eine kurze Beschreibung. Diese werden über den

[4] NASA Task Load Index: humansystems.arc.nasa.gov/groups/TLX/
[5] System Usability Scale: measuringu.com/sus/

REST-Endpoint des Repositorys abgerufen und addieren maximal noch wenige Millisekunden zur insgesamt wahrgenommenen Verzögerung der Vorschlagsliste im Assistenten.

Bei den *Indikatoren für die Erfüllungsgrade von Bedingungen* wurde zunächst die Zweiteilung abgeschafft, um die räumliche Verteilung Awareness-Visualisierung zu vermeiden, die sich während der Nutzerstudie als ungünstig herausgestellt hat. Dadurch entstand jedoch die Herausforderung, die Gesamtbewertung und die Teilbewertungen kompakt an einer Stelle darzustellen. Schließlich wurde auf eine explizite Darstellung der Gesamtindikatoren verzichtet und die kompositen Bewertungen innerhalb des Bedingungsbaumes in den Junktor-Boxen dargestellt. Die Gesamtbewertung mit dem Anteil der Übereinstimmung anderer Kriterien als die der Qualitätsanforderungen wird nun nur noch implizit durch die Position in der sortierten Liste der vorgeschlagenen Kompositionsfragmente angezeigt. Dem Nutzer ist zuzutrauen, dass er die nach oben sortierten Vorschläge als die mit dem höchsten Gesamterfüllungsgrad identifiziert. Durch die kompaktere Darstellung der Bedingungen in Boxen, aus deren Anordnung auch die Struktur der Komposition hervorgeht, wird weniger Platz geboten, der für die Darstellung der Indikatoren der Erfüllung genutzt werden kann. Eine textbasierte Anzeige – vgl. Prozentangaben rechts in Abbildung 7.11 – ist dadurch von der Platzaufteilung her problematisch. Hier müsste außerhalb dynamisch Platz geschaffen werden. Deshalb wurden Varianten mit Overlays in die nähere Betrachtung gezogen. Abbildung 7.12 zeigt eine vergleichende Darstellung der Indikatoren für die Bedingungen in einem Bedingungsbaum einer Qualitätsanforderung. Die obere linke Darstellung zeigt einen Bedingungsbaum mit Boxen ohne Indikatoren der Erfüllungsgrade. Die anderen drei Darstellungen zeigen unterschiedliche Visualisierungen derselben Auswertungsergebnisse für eine komposite Bedingung aus einer Fuzzy- (oben) und einer diskreten Bedingung (unten).

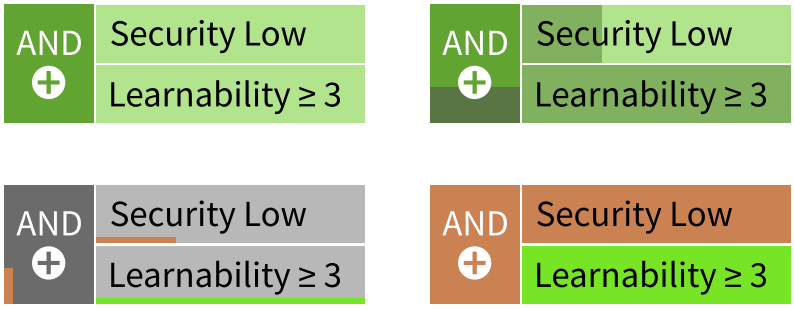

Abbildung 7.12: Vergleich von Indikatoren für den Erfüllungsgrad von Bedingungen

Oben rechts ist die in der Referenzimplementierung bevorzugt genutzte Variante mit Hintergrundstreifen als Indikatoren zu sehen. Dabei wird die Bedingungsbox und auch die Box des Junktors stellvertretend für die komposite Bedingung zu dem Anteil des jeweiligen Auswertungsergebnisses mit einer Highlight-Farbe ausgefüllt. Vorteil davon ist, dass die Darstellung gut ablesbar ist, jedoch die Erkennbarkeit des darüberliegenden Textes nicht zu stark beeinträchtigt. Unten links ist eine alternative Darstellung mit schmalen Streifen zu sehen. Hierbei wurde die Auswertungsinformation redundant über die Streifenlänge und die Streifenfarbe (auf einer Skala von rot bis grün) visualisiert. Für eine bessere Ablesbarkeit der Farbstreifen wurde die Hintergrundfarbe der Bedingungsboxen grau gewählt. Unten rechts fällt die Streifenlänge als Kriterium weg. Dafür wurde die ganze Box jeweils in der Farbe des Erfüllungsgrades einge-

färbt. Diese sehr auffällige Darstellung hat jedoch den Nachteil, dass die Schrift bei bestimmten Farbkombinationen schlechter lesbar ist. Durch fehlende Referenzfarben ist hierbei der genaue Erfüllungsgrad auch schlechter ablesbar. Eine zusätzliche Farblegende, um die Ausprägungen der Streifen für die Kandidaten besser einschätzen zu können, würde zusätzlichen Platz verbrauchen. Die Variante oben rechts wurde nach einer Experteneinschätzung daraufhin als neue Standardform der Indikation der Erfüllungsgrade beim Hovern über einen Kandidaten gewählt. Für die Indikatorstreifen in den Boxen der Junktoren ist sowohl eine horizontale Ausdehnung als auch eine vertikale Ausdehnung möglich, wie sie in Abbildung 7.5 zu sehen ist.

7.8 Diskussion zur Validierung und Implementierung

Ziel der Validierung war es, die Umsetzbarkeit und Akzeptanz der Konzepte der Arbeit innerhalb der Szenarien zu überprüfen. Dabei wurde insbesondere auf die Nutzerbeteiligung in den Anwendungsfällen und Testdaten Wert gelegt, die in den Szenarien ① und ② eine Rolle spielen. Die Grundlage dafür bietet die umfassende Implementierung der Mashup-Infrastruktur, die um Werkzeuge, Funktionen und Prozesse der qualitätsgetriebenen Spezifizierung und automatisierten Auswertung von Anforderungen erweitert wurde. Der Anforderungsassistent stellt dabei das wichtigste Werkzeug dar. Er wurde über mehrere Iterationen der Laufzeitumgebung fortentwickelt und setzt Feedback aus dem Verhalten und den Anmerkungen einer Nutzerstudie um, die vor allem die Umsetzung der spezifizierbaren Elemente des Modells für Qualitätsanforderungen auf Gebrauchstauglichkeit bewertet hat. Dabei wurde insbesondere überprüft, wie gut die Nutzerintention durch ein solches Werkzeug erfassbar ist, um im nächsten Schritt vollautomatisiert weiterverarbeitet zu werden.

These 1 fordert den einheitlichen Zugriff auf qualitätsrelevante Basiswerte über eine geeignete Datenmodellierung. Dies konnte durch die Implementierung des Modells für Qualitätseigenschaften erreicht werden. Insbesondere die konzeptionelle Unterscheidung der Typen der Wertermittlung führt hierzu eine neue Strukturierung der Eigenschaften ein. Zu jedem Teilaspekt wurde eine konkrete Infrastruktur geschaffen. Dabei handelt es sich um die direkte Modellierung statisch festgelegter Eigenschaftswerte für Metadaten innerhalb der *semantischen Modelle der Kompositionsfragmente*, die exemplarisch implementierte *Sensorik* für ausgewählte Eigenschaften und die *Infrastruktur zur Verwaltung gesammelter Werte*. Eine zusätzlich geschaffene Abstraktion auf *Features*, die auch anwendungsspezifische Nutzdaten innerhalb von Qualitätsanforderungen zugänglich machen, ermöglicht Laufzeittests und Plausibilitätsprüfungen, die Gegenstand von Szenario ② sind. Durch den in mehreren Beispielszenarien praktisch erprobten, einheitlichen Zugriff auf diese Features über eine einheitliche Zugriffsschicht für Qualitätsanforderungen gilt These 1 als erfüllt.

Zur Unterstützung des iterativen Vorgehens bei der Live-Sophistication fordert These 2 die automatisierte Auswertung von Qualitätsanforderungen. Dadurch werden in Verbindung mit Empfehlungsprozessen kurze Iterationszyklen bei der Entwicklung erreicht. In Konsequenz erhöht die automatisierte Auswertung die durch den Nutzer angestrebte Adaptivität und unterstützt die iterative Entwicklung. Das Ziel der automatisierten Auswertung wird in erster Linie durch die *maschinenlesbare Modellierung anpassbarer Qualitätsanforderungen* erreicht. Unabhängig von ihrer Herkunft – durch das nutzergetriebene Bearbeiten im Anforderungsassistent, das Laden favorisierter Anforderungen oder das Importieren von Anforderungsmodellen

– kann die Anforderungsauswertung vollautomatisch erfolgen. Die getestete Performance im Auswertungsprozess stellt sicher, dass es in der iterativen Weiterentwicklung nicht zu ungewünschten Verzögerungen kommt, vgl. Abschnitt 7.7. Ob umgangssprachliche Formulierungen sinnvoll in maschinenlesbare Qualitätsanforderungen transformiert werden können, war Untersuchungsgegenstand der Nutzerstudie. These 3 fordert dies, um das Spezifizieren von Qualitätsanforderungen für Web-Nutzer ohne Programmiererfahrung zu erleichtern. Als Beitrag dieser Arbeit konnte ein Konzept zur Verwertung von Fuzzy-Termen konzipiert und umgesetzt werden. Das Verwenden dieser Fuzzy-Terme erleichtert das Zuordnen unscharfer Forderungen für den Nutzer. Ein darüber hinausgehendes Konzept für eine umfassende Unterstützung von natürlicher Sprache zur Anforderungsgewinnung wurde in dieser Arbeit nicht betrachtet. Da der Vorgang der Funktionsanpassung für Fuzzy-Terme als kritisch bewertet wurde, ist er nicht mehr Teil des aktuellen Anforderungsassistenten. Administrativ können diese Funktionen abseits der Mashup-Laufzeitumgebung in einem Fuzzy-Set-Editor definiert und angepasst werden.

Qualitätsanforderungen werden unabhängig vom konkreten Anwendungsszenario mit einem bestimmten Zweck festgelegt. Das Erfüllen dieses Zwecks und damit das *Sicherstellen der Wirksamkeit* ist Gegenstand von These 4, indem Aktionen an Qualitätsanforderungen gekoppelt werden. Diese Arbeit stellt ein Konzept für Aktionen vor, die erst die Adaptivität der komponierten Anwendungen ermöglichen. Dabei wurde darauf geachtet, dass die typischen Entwicklungsaktionen, beispielsweise der Komponentenaustausch, und weitere Aktionen unter Einbeziehung von Nutzerfeedback angeboten werden. These 4 wird somit durch das Umsetzen der Beispielszenarien validiert, indem gezeigt wird, dass die Handlungsvorschriften den gewünschten Zweck durchsetzen. Zum Erhöhen der Effizienz geplanter Entwicklungsaufgaben wurde die Nutzung *anpassbarer Qualitätsprofile* eingeführt, vgl. These 5. So muss der Nutzer nicht jedes Mal von Neuem mit der Komposition von Anforderungen und insbesondere den damit verbundenen Bedingungsbäumen beginnen. Eine explizite Zeitmessung zur Qualtifizierung der Effizienz wurde hierbei nicht durchgeführt. Außerdem sollen *wiederverwendbare Favoriten* die anforderungsgetriebene Entwicklung weiter vereinfachen.

Wie andere webbasierte und vor allem verteilte Laufzeit-Plattformen, gelten auch für die hier konzipierte und innerhalb der Plattform CRUISE implementierte Infrastruktur Randbedingungen, die eine *Abhängigkeit von der Entwicklung der Browser-Technologien und Web-Standards* erzeugen. Neben dem Grundproblem der Nicht-Erreichbarkeit von Diensten muss Wartungsaufwand in die Etablierung von Sicherheitsmaßnahmen wie der Transportverschlüsselung betrieben werden. Über die Betrachtung der Komplexität der Vorgänge mit Nutzerbeteiligung und der Messung der Durchlaufzeiten hat sich herausgestellt, dass die für den Nutzer relevanten Verzögerungen, die durch Netzwerklatenzen und die Dauer von Berechnungen entstehen, keine Beeinträchtigung der Benutzbarkeit der Mashup-Plattform allgemein und der Werkzeuge innerhalb der Qualitätsinfrastruktur im Besonderen darstellen. Eine potenzielle Schwäche der Nutzung des anforderungsgetriebenen Empfehlungssystems besteht darin, dass die Empfehlungsalgorithmen nicht wie vom Nutzer erwartet oder sogar fehlerhaft arbeiten. Dies macht sich insbesondere bemerkbar, wenn die Auswertungsergebnisse für Qualitätsanforderungen dieser Arbeit in ein Empfehlungssystem integriert werden, das weitere Parameter wie die Nutzungshäufigkeit oder kollaboratives Filtern berücksichtigt, wie es in [Rad+12] angestrebt wird.

Kapitel 8

Zusammenfassung, Diskussion, Bewertung und Ausblick

Diese Arbeit schlägt anhand neuartiger Konzepte den qualitätsbewussten Umgang mit kompositen Web-Mashups vor, indem sowohl die Nutzung als auch die Entwicklung mit umfassender Modellierungsgrundlage und Auswertungsinfrastruktur für anpassbare Qualitätsanforderungen in einem Kontinuum an Prozessen verbessert wird. Nach einer Zusammenfassung der bisherigen Kapitel werden die wichtigsten Forschungsbeiträge diskutiert und bewertet. Dabei steht die Gegenüberstellung der zu erreichenden Ziele – in Verbindung mit den initial vorgestellten Anwendungsszenarien – mit der Validierung der entwickelten Konzepte im Vordergrund. Über den erreichten Mehrwert hinausgehend werden schließlich Anknüpfungspunkte zu aktuellen und möglichen künftigen Arbeiten in einem Ausblick beschrieben.

8.1 Zusammenfassung der Kapitel

Nach der initialen Motivation und Zielstellung in Kapitel 1 folgen wichtige Grundlagen, Anwendungsszenarien und Begriffsabgrenzungen in Kapitel 2 sowie die Einordnung in existierende Arbeiten in Kapitel 3. Im anschließenden konzeptionellen Teil wird zunächst die Modellierungsgrundlage für Qualitätseigenschaften in Kapitel 4 sowie für Qualitätsanforderungen in Kapitel 5 samt Metamodell und Referenzinstanzen vorgestellt. Kapitel 6 ordnet die Modelle und Vorgänge in einen umfassenden Entwicklungsprozess für Mashups ein und stellt mögliche Adaptionsaktionen vor. Implementierung und Validierung der präsentierten Konzepte unterstreichen in Kapitel 7 die Eignung der Modelle, Konzepte und Prozesse innerhalb der vorgesehenen Anwendungsszenarien. Die Kapitel beinhalten im Einzelnen:

Kapitel 1 · Motivation und Zielstellung

Aus dem Grundbedürfnis der situativen Erstellung und Nutzung kompositer Web-Mashups wurden zunächst die Hauptprobleme und daraus die Thesen und Zielstellung dieser Arbeit abgeleitet. Da sich über das Mashup-Paradigma mittlerweile komplexe und leistungsstarke Anwendungen erstellen bzw. betreiben lassen, wird vorgeschlagen, mit der Nutzung von Qualitätseigenschaften eine anforderungsgerechte Auswahl von Kompositionsfragmenten zu realisieren. Um nicht nur das Empfehlen, sondern auch das Ausführen in den Kontext der Qualitätsanforderungen zu stellen, fordert dieses erste Kapitel eine einheitliche Modellierungslandschaft für den Zugang auf qualitätsrelevante Basiswerte innerhalb verschiedener Anwendungsfälle. Aus Nutzersicht steht dabei das möglichst einfache Festlegen solcher Anforderungen im Vordergrund.

Deshalb werden Techniken für die erleichterte Erfassung der Nutzerintention mit umgangssprachlichen Ausdrücken gefordert. Neben der automatisierten Auswertung der so generierten Anforderungen und der Kopplung an Handlungsvorschriften steht die Integration in einen qualitätsbewussten Entwicklungsprozess als anzustrebendes Ziel im Fokus. Damit soll außer der empfehlungsgetriebenen Entwicklung auch die Überwachung von Qualitätsanforderungen zur Laufzeit kompositer Anwendungen und die damit verbundene Adaptivität erreicht werden.

Kapitel 2 · Grundlagen der Entwicklung und Nutzung kompositer Web-Mashups

Das Grundlagenkapitel beschäftigt sich zuerst mit der Abgrenzung der Charakteristika des Anwendungstyps Web-Mashup. Er wird dabei vor allem im Kontext der serviceorientierten Architektur, der kompositen Softwaresysteme und der Web-Anwendungen betrachtet. Da die Motivation der Einführung von Qualitätsanforderungen stark durch den Entwicklungsprozess geprägt ist, werden anschließend typische Entwicklungsmethoden für das Mashup-Paradigma abgegrenzt, woraus Nutzungs- und Entwicklungsszenarien abgeleitet werden, auf die in der ganzen Arbeit immer wieder Bezug genommen wird. Ebenfalls abgegrenzt wird die Zielgruppe bzw. die Rollen, die an den vorgestellten Entwicklungsmethoden beteiligt sind. Schließlich folgt eine Definition bzw. Interpretation der wichtigsten Qualitätsbegriffe im Kontext dieser Arbeit. Sie dienen als Ausgangspunkt für die Einordnung vorhandener Forschungsarbeiten und Standards im darauffolgenden Kapitel.

Wichtige Forschungsbeiträge: Die in der gesamten Arbeit referenzierten Anwendungsszenarien umfassen die Live-Sophistication, also das iterative Erstellen bzw. Verbessern einer Mashup-Anwendung durch das Durchführen von Entwicklungsaufgaben mit möglichst nahtloser Integration in die Nutzung, das Überwachen von Qualitätsanforderungen fertiger Anwendungen zur Laufzeit sowie das Hinterlegen modellierter Anforderungen in einem Profil. Obwohl das nutzergetriebene Vorgehen bei der Mashup-Erstellung auch durch andere Forschungsarbeiten thematisiert wird, ist das anforderungsgetriebene Vorgehen mit automatisierter Auswertung in den definierten Szenarien ein neuartiger Untersuchungsgegenstand. Dadurch wird deutlich, dass die Modellierungs- und Laufzeitinfrastruktur sowohl eine einheitliche Abstraktion auf heterogene Basisdaten als auch die Unterstützung für unterschiedliche Anwendungsszenarien mit verschiedenen Zielgruppen bieten muss.

Kapitel 3 · Stand der Forschung und Technik

Vor allem die Modellierung von Qualitätseigenschaften und -anforderungen wurde von einigen Standards beeinflusst, die den ersten Teil dieses Kapitels einnehmen. Dort wird vor allem auf die Strukturierung und Abstraktion der Modelle eingegangen. Anschließend gibt das Kapitel einen kurzen Überblick der Arbeiten zur Aufgabenmodellierung und Fuzzy-Sets als Vorbereitung der Konzepte zur Anforderungsgenerierung mit umgangssprachlichen Termen. Das restliche Kapitel nimmt Stellung zu Forschungsarbeiten, die nach fünf Kriterien in Clustern untersucht wurden, welche die wichtigsten Typen von Anwendungssystemen, die mit Web-Mashups verwandt sind, abdecken. Die Erkenntnisse werden schließlich kompakt zusammengefasst.

Wichtige Forschungsbeiträge: Neben den Erkenntnissen der vorhandenen Standards zum Aufbau von Modellen für die Produktqualität und generellen Kriterien in der Qualitätsmodellierung wird in der Cluster-weisen Untersuchung deutlich, dass bei Web-Anwendungen und insbesondere bei Web-Mashups Qualitätseigenschaften in mehreren Communitys grundlegend behandelt wurden, jedoch die Anforderungsmodellierung und damit auch die Anforderungsauswertung keine wesentliche Rolle spielt. Dieser Aspekt ist viel besser bei Web-Services und in der komponentenbasierten Softwareentwicklung erforscht, die jedoch die Eigenschaftsträger in einer anderen Granularität und UI-spezifische Elemente überhaupt nicht betrachten. Eine im Kontext von Web-Mashups verwendbare Infrastruktur zur automatisierten Auswertung von Qualitätsanforderungen wurde somit als große Schwäche existierender Arbeiten herausgestellt. Bereits bei der Modellierung maschinenlesbarer Anforderungen existieren keine Forschungsansätze, die ohne Weiteres auf die geplante Mashup-Infrastruktur anwendbar wären. Allerdings schlagen mehrere Arbeiten Mengen von Qualitätseigenschaften vor, die in ihrer Gesamtheit in das Referenzmodell dieser Arbeit einbezogen werden können. Ebenfalls gemischt fällt die Bewertung der Integration in Entwicklungsumgebungen und -prozesse aus. Einzelne Arbeiten beschäftigen sich mit der nutzergetriebenen Entwicklung, stehen jedoch mit ihren Empfehlungssystemen abseits der Komplexität der hier betrachteten Qualitätsanforderungen. Die geforderte Kopplung von Handlungsvorschriften an Qualitätsanforderungen ist ein Konzept, das nur in Ansätzen in der komponentenbasierten Softwareentwicklung existiert. Hier besteht großes Potenzial, um ein Alleinstellungsmerkmal durch die umfassende Unterstützung verschiedener anforderungsgetriebener Anwendungsfälle, vor allem mit Adaptivität, zu etablieren.

Kapitel 4 · Modellierung von Qualitätseigenschaften für Mashups

Dieses Kapitel beschreibt die Modellierung von Qualitätseigenschaften als erstes konzeptionelles Ergebnis dieser Arbeit. Dabei wird zunächst erläutert, wie verschiedene Teile der umfassenden Modellierungslandschaft zusammenhängen und ggf. Drittmodelle integrieren. Anschließend werden spezifische Anforderungen an die Modellierung sowie die Modellstruktur in einem Metamodell vorgestellt. Nach einem detaillierten Einblick in die einzelnen Charakteristika beinhaltet ein weiterer Teil des Kapitels das Referenzmodell für Qualitätseigenschaften, das die strukturellen Vorschläge anhand einiger typischer Qualitätseigenschaften für Web-Mashups instanziiert. Dabei wird Bezug auf die in existierenden Arbeiten vorgestellten Eigenschaften – beispielsweise die Modelle von Cappiello, Daniel und Matera aus [CDM09] und [Cap+11a] – genommen und deren Eignung für die hier konzipierte Mashup-Plattform bewertet.

Wichtige Forschungsbeiträge: Konzeptionelle und gleichzeitig praktische Ergebnisse dieses Kapitels sind das Metamodell für Qualitätseigenschaften und das dazu gehörende Referenzmodell. Dabei wurden zahlreiche strukturelle Charakteristika eingeführt. Hierzu zählen vor allem die Möglichkeit der Zuordnung von Fuzzy-Termen und Fuzzy-Mengen zu den Eigenschaften, die Facettierung in Concerns – ebenfalls mit einem Referenzmodell – und die Unterteilung in Typen der Ermittlung von Werten für die Eigenschaften. Zudem wurde die Aggregation von Eigenschaften auf Anwendungsebene konzipiert, die Voraussetzung für den später vorgestellten Anwendungsbrowser ist. Durch die Nachweise der Verwendung für bestimmte Einträge des Referenzmodells in anderen Forschungsarbeiten ist zudem ein Nachschlagewerk für Qualitätseigenschaften bei Mashups mit der Angabe von Metriken, Datentypen und Einheiten entstanden.

Eine weitere konzeptionelle Besonderheit des Eigenschaftsmodells ist es, kaum veränderliche *Metadaten* analog zu Qualitätseigenschaften zu behandeln, da diese aus Sicht der Nutzeranforderungen genauso verwendet werden können.

Kapitel 5 · Festlegen und Auswerten von Qualitätsanforderungen

Dank der einheitlichen Modellierungsgrundlage, die in Kapitel 4 vorgestellt wurde, beschäftigt sich dieses Kapitel mit der Modellierung, Eingabe und Auswertung von Qualitätsanforderungen. Dabei wird zunächst eingeordnet, wie Qualitätsanforderungen in der Mashup-Infrastruktur genutzt werden können und welche Herausforderungen sich aus ihrer Nutzung ergeben. Anschließend wird das Metamodell für Qualitätsanforderungen präsentiert sowie das nutzergetriebene Erzeugen und die automatisierte Auswertung beschrieben.

Wichtige Forschungsbeiträge: Das Modell für Qualitätsanforderungen kann als einheitliche Formalisierung in allen drei Beispielszenarien genutzt werden. Die Herausforderungen beim Erstellen von Anforderungsinstanzen werden dabei auf den Empfehlungs- und den Überwachungsmodus aufgeteilt. Das Metamodell bezieht sich dabei auf *Features*, die entweder im Modell für Qualitätseigenschaften definiert wurden oder aus anderen Eigenschaften von Kompositionsfragmenten stammen und beispielsweise Zustände der Anwendungskomponenten zugänglich machen. Die strukturellen Charakteristika umfassen dabei vor allem den Einsatz der Entwurfsmuster *Event Condition Action (ECA)* und *Composite*. Ebenfalls werden diskrete, Fuzzy- und kombinierte Bedingungen unterstützt. Für die gekoppelten Events wird ein Referenzmodell vorgeschlagen, das in der Implementierung den Überwachungsmodus und dessen Automatisierung ermöglicht. Mit diesem Konzept wird die Lücke bestehender Arbeiten geschlossen, um eine automatisierte Anforderungsauswertung in einer Mashup-Plattform zu gewährleisten. Gleichzeitig werden verschiedene Arten zur umfassenden Anforderungsgewinnung betrachtet, die innerhalb der Beispielszenarien gefordert sind.

Kapitel 6 · Qualitätsbewusster Entwicklungs- und Nutzungsprozess

Dieses Kapitel fasst die vielfältigen Verwendungsmöglichkeiten einer Mashup-Plattform zu einem umfassenden Entwicklungs- und Nutzungsprozess zusammen. Aufbauend auf diesem Prozess werden die Verbesserungen aufgezeigt, die durch die Infrastruktur der Qualitätsmodellierung und -auswertung entstehen. Der zweite Teil des Kapitels beschäftigt sich mit den Handlungsvorschriften, die unter anderem die Adaptivität der Mashup-Anwendungen ermöglichen. Dafür wird zum einen ein Referenzmodell präsentiert, dessen Aktionen innerhalb von Anforderungsinstanzen genutzt werden können. Zum anderen wird auf den Lebenszyklus und mögliche Konflikte beim Aufeinandertreffen mehrerer Aktionen eingegangen.

Wichtige Forschungsbeiträge: Das Kapitel beinhaltet zwei Hauptergebnisse. Im ersten Teil wird der integrierte Entwicklungs- und Nutzungsprozess für komposite Web-Mashups beschrieben, auf Grundlage dessen die Verbesserungen durch die Qualitätsinfrastruktur aufgezeigt werden. Darin werden unter anderem die Konzepte der Live-Sophistication, weiterer Einstiegspunkte in die Mashup-Laufzeitumgebung sowie die Zusammenhänge der Entwicklungsaktionen systematisiert. Die Verbesserungen für den Einstieg in die Entwicklungstätigkeit mit der Plattform

betreffen das Ableiten von Nutzerintentionen mit Hilfe von Qualitätsanforderungen und das verbesserte Abbilden, d. h. Importieren und empfehlungsbasiertes Zuordnen, von Drittmodellen mit Qualitätsanforderungen auf Kompositionsvorschläge. Während der integrierten Nutzung und Entwicklung wird eine neue Art der Laufzeitadaptivität – beispielhaft durch die Überwachung von Qualitätseigenschaften gezeigt – und das automatisierte Auslösen von Entwicklungsschritten erreicht. Diese Entwicklungsaktivitäten sind Teil des ebenfalls in diesem Kapitel erarbeiteten Modells für Aktionen. Diese werden in einen Lebenszyklus von Anforderungen eingeordnet und mit einem Referenzmodell illustriert. Detailliert wird auf das Konfliktpotenzial zeitnah auftretender Aktionen eingegangen, indem Präventionsstrategien und Konsequenzen diskutiert werden.

Kapitel 7 · Validierung und Implementierung

Zu Beginn des Validierungskapitels wird methodisch unterschieden, welche Ergebnisse unter Nutzung bestimmter Techniken validiert wurden. Dabei wird zunächst ein Überblick der implementierten Infrastruktur gegeben. Nach der Beschreibung der Referenzarchitektur, der Beispieldaten für umgesetzte Kompositionsfragmente und der Werkzeuge werden Aufbau und Ergebnisse von Nutzerstudien bzw. einzelne Ergebnisse, die durch Experteneinschätzungen entstanden sind, beschrieben. Außerdem erfolgt eine Diskussion der Verbesserungen in den Werkzeugen, die als Entwurfsentscheidungen und Konsequenz der Nutzereinschätzungen zum Erreichen der geforderten Forschungsziele gefällt wurden.

Wichtige Forschungsbeiträge: Die Forschungsergebnisse, die als Modelle, Algorithmen, Funktionen, Dienste und Prozesse entstanden sind, konnten als Referenz-Infrastruktur umgesetzt werden. Die technologische Grundlage ist dabei die Mashup-Plattform CRUISE, die mit dieser Arbeit substanziell weiterentwickelt wurde. Eine Übersicht grenzt die konkret erweiterten und neu hinzugekommene Bestandteile nach Schichten sortiert gegen die bestehende Ausstattung der Plattform ab. Das Ziel der verbesserten Nutzung der Plattform durch den Menschen wurden durch die Umsetzung der Werkzeuge, vor allem durch den Anforderungsassistenten, erreicht. Der Umgang mit diesen Werkzeugen und damit auch der Einsatz bestimmter Konzepte konnte durch die Erprobung bzw. Einschätzung von Nutzern und Experten validiert und optimiert werden. Die konkreten Beiträge der Validierung und Implementierung zu den einzelnen Thesen und Forschungszielen werden am Ende des Validierungskapitels zusammen mit den Rahmenbedingungen und Risiken, die sich aus der Anwendung der vorgeschlagenen Konzepte und Plattformerweiterungen ergeben, diskutiert.

8.2 Diskussion und Bewertung der Forschungsergebnisse

Die erreichten Forschungsergebnisse werden nun den Anforderungen aus dem ersten und zweiten Kapitel gegenübergestellt. Dabei wird insbesondere überprüft, inwieweit die Aussagen der Forschungsthesen, siehe Abschnitt 1.2, valide sind. Außerdem wird die Eignung sowie die Abdeckung der Anwendungsszenarien durch die erarbeiteten Konzepte und die vorgenommene Implementierung diskutiert.

8.2.1 Modellierungslandschaft

In These 1 wird gefordert, dass qualitätsrelevante Basisdaten über eine einheitliche Schnittstelle nutzbar gemacht werden. Dies ist gelungen, indem ein Metamodell für Qualitätseigenschaften sowohl die heterogene Befüllung der Eigenschaften mit Werten als auch die einheitliche Repräsentation – vor allem im Kontext der maschinellen Verarbeitung – regelt. Die jeweils geeignete Infrastrukturkomponente ist damit in der Lage, Werte für Qualitätseigenschaften bereitzustellen, egal ob sie zur Laufzeit gemessen, vom Autor eines Kompositionsfragmentes bereitgestellt oder historisch bzw. über die Einzelbewertungen vieler Nutzer aggregiert wurden. Die Modellierungslandschaft wurde dabei durch den Einsatz semantischer Modellierungssprachen so konzipiert, dass Drittmodelle sehr einfach angebunden werden können, wie es beispielsweise mit QUDT geschehen ist. Teil dieser Modellierungslandschaft ist insbesondere auch das Modell für Qualitätsanforderungen, das sowohl in der Lage ist, über die implementierte Werkzeugunterstützung Anforderungen von Nutzern entgegenzunehmen, als auch Anforderungen aus Fremdmodellen zu importieren. Durch die Modellierung mit RDF bzw. OWL wurde vor allem die Modellierungsanforderung der Erweiterbarkeit umgesetzt. Zudem konnte eine breite Unterstützung durch Frameworks für die Wissensmodellierung wie Apache Jena und insgesamt eine sehr gute Integration in die genutzte Referenzplattform CRUISE erzielt werden.

8.2.2 Automatisierte Anforderungsauswertung

Basis für die automatisierte Anforderungsauswertung, die in These 2 gefordert wird, ist das Modell für Qualitätsanforderungen. Es ist darauf ausgelegt, dass die Wertebelegung über Sensoren und Autorenwerkzeuge einer einheitlichen Repräsentation der referenzierten Eigenschaften zugeführt werden kann. Durch das Verwenden des Musters Event Condition Action (ECA) aus regelbasierten Systemen kann die Anforderungsauswertung mit den hinterlegten Events alle Beispielszenarien abdecken. Dies beinhaltet in erster Linie die Empfehlung im Kompositionsprozess, in dem Anforderungen iterativ geändert werden und die Auswertungsergebnisse schnell zur Verfügung stehen müssen. Auf der anderen Seite wird das vollautomatische Überwachungsszenario umgesetzt, das vordefinierte Qualitätsanforderungen zur Laufzeit von Web-Mashups akzeptiert. Insbesondere hier ist das benutzerdefinierte Festlegen geeigneter Events zum Auslösen des Auswertungsprozesses erforderlich. Durch dieses Szenario wird in Kopplung mit Aktionen die Adaptivität kompositer Anwendungen in der Plattform ermöglicht. Das dritte Szenario verwendet dieselben Techniken, lässt jedoch das Hinterlegen von Anforderungen in Kompositionsfragmenten und anderen Profilen zu. Mit der verteilten Wertermittlung, die für den Vergleich im Auswertungsalgorithmus benötigt wird, ist ein hybrider Auswertungsprozess entstanden, der Zugriff auf alle Arten von Eigenschaften im Referenzmodell bietet. Die in These 2 angestrebte Adaptivität auf Grundlage der maschinenverarbeitbaren Anforderungen wurde somit ebenfalls erreicht.

8.2.3 Spezifikation mit umgangssprachlichen Begriffen

Bereits bei der Modellierung wurde auf das Ziel hingearbeitet, Nutzern das Spezifizieren von Qualitätsanforderungen zu erleichtern, indem ein Zugang für das Verwenden umgangssprachlicher Begriffe entstanden ist, der in These 3 gefordert wird. Als Forschungsergebnis konnte hierfür das Konzept der Fuzzy-Mengen auf den Erfüllungsgrad von Anforderungen für jeweils

eine bestimmte Qualitätseigenschaft projiziert werden. Die zu den Funktionen zugeordneten Fuzzy-Terme repräsentieren dabei die im Kontext einer Eigenschaft anwendbaren Begriffe wie *günstig* für den Preis oder *möglichst niedrig* für den Energieverbrauch. Mit der Fuzzy-Logik ist es zudem möglich, diskrete und Fuzzy-Bedingungen in einem Bedingungsbaum zu verwenden und trotzdem ein skalares Auswertungsergebnis zu erhalten, das zwingend im Empfehlungsprozess benötigt wird, um das Bilden einer Reihenfolge zu ermöglichen. Die Labels der Terme sind abhängig von der natürlichen Sprache und ggf. vom soziokulturellen Kontext des Nutzers, der verantwortlich für das Erstellen der Anforderungen ist. Durch Internationalisierung bzw. Lokalisierung kann diese Hürde umgangen werden. Dies wird jedoch nicht in dieser Arbeit betrachtet. Eine Herausforderung stellt dabei allerdings das automatisierte Ableiten von Anforderungen aus Fremdtexten dar, die möglicherweise sprachlich gemischte Inhalte aufweisen. Diese umfassende Unterstützung natürlicher Sprache als Anforderungsquelle stand ebenfalls nicht im Fokus, da im Entwicklungsprozess eine interaktive Anforderungsdefinition angestrebt wird. Eine weitere Erleichterung, die in dieser Arbeit umgesetzt wurde, ist die Profilbildung über Qualitätsanforderungen. Dies ist ein typischer Ansatz zu Komplexitätsverlagerung. Jedoch steht es dem Nutzer frei, die vorkonfigurierten Profile selbst anzupassen. Zusätzliche Hilfe bietet auch die Möglichkeit, Anforderungen als Favoriten zu speichern und wiederzuverwenden.

8.2.4 Überwachung und Durchsetzung

Der Aktionsteil im ECA-Muster einer Qualitätsanforderung ist verantwortlich für das Auslösen vorbestimmter Handlungsvorschriften und damit die Anforderungsdurchsetzung, die Gegenstand von These 4 ist. Das Referenzmodel für Aktionen bietet dabei eine große Anzahl von Adaptions-, Awareness- und Interaktionsmöglichkeiten. Beispielsweise kann entschieden werden, dass ein vollautomatischer Komponentenaustausch erfolgt, sobald die gekoppelte Bedingung verletzt wird. Auch das simple Anlegen von Log-Einträgen oder das Anfordern von Expertenunterstützung ist über die in dieser Arbeit vorgestellten Aktionsinfrastruktur möglich. Das Betrachten mehrerer zeitnah auftretender Aktionen in einer Konfliktmatrix zeigt, dass in den meisten Fällen keine besondere Behandlung notwendig ist. Viele potenzielle Probleme entstehen ohnehin nur auf der Nutzerseite im zeitlichen Zusammenhang, beispielsweise bei mehreren sich überlagernden Modaldialogen oder Hinweisfenstern. Sie werden aus Sicht der Mashup-Plattform als konkurrierende Standardfälle nicht als Problem wahrgenommen. Hierzu könnte eine empirische Analyse bestimmter Kombinationen von Aktionen die Konfliktbetrachtung weiterführen. Dafür ist jedoch der umfassende Aufbau weiterer Beispielszenarien mit reproduzierbaren Anforderungen und deren zeitlicher Abfolge notwendig. Ein derart komplexes Aufeinandertreffen von Adaptionsaktionen ist nicht Teil der betrachteten Beispielszenarien, sodass die Anforderung des Sicherstellens der Wirksamkeit von Qualitätsanforderungen aus These 4 als erfüllt betrachtet wird.

8.2.5 Integrierter Entwicklungs- und Nutzungsprozess

Da in existierenden Forschungsarbeiten kein grundlegender Prozess für die integrierte Entwicklung und Nutzung kompositer Web-Mashups definiert wurde, stellt ein systematischer Entwicklungsprozess mit situationsabhängigen Einstiegspunkten das erste Ergebnis dieser Arbeit zu These 5 dar. Er setzt unter anderem die im Kontext dieser Forschungsarbeit vorgeschlagene

Live-Sophistication um. Auf Basis dieses Prozesses werden die Assistenzfunktionen, die durch die Qualitätsinfrastruktur den Entwicklungs- und Nutzungsprozess verbessern, vorgestellt. Die für den Nutzer in erster Linie sichtbare Verbesserung ist dabei der Anforderungsassistent als Werkzeug. Er tritt in den beiden Beispielszenarien mit menschlicher Beteiligung beim Festlegen der Qualitätsanforderungen in Erscheinung. Durch die Nutzer- und Expertenstudien konnten die verwendeten Interaktionskonzepte validiert bzw. optimiert werden, sodass die erleichterte Abarbeitung der untersuchten Aufgaben ermöglicht wird. Für die in These 5 geforderte Erhöhung der Effizienz beim Bearbeiten von Entwicklungsaufgaben wurden die Qualitätsprofile als Bausteine für Qualitätsanforderungen mit vorgefertigten Bedingungsbäumen sowie die Möglichkeit des Setzens von Favoriten konzipiert und implementiert. Als quantitativer Beleg für die Verbesserung des Entwicklungsprozesses wäre eine vergleichende Zeitmessung denkbar, die im Rahmen dieser Arbeit nicht durchgeführt wurde. Das Verbesserungspotenzial im Entwicklungsprozess ist natürlich abhängig vom individuellen Nutzerverhalten, sodass hier nur eine schwache Indikation gegeben werden kann.

8.3 Ausblick auf aktuelle und künftige Arbeiten

Wie die Referenzinfrastruktur dieser Arbeit durch die Ergebnisse aus der CRUISE-Plattform inspiriert wird, liefert sie auch Beiträge für parallele Forschungsarbeiten. Diese Ansatzpunkte werden nun kurz vorgestellt. Schließlich wird ein Ausblick auf mögliche Untersuchungsgegenstände gegeben, die nicht in aktuellen Arbeiten behandelt werden, jedoch interessante bzw. lohnenswerte Forschungsgegenstände darstellen.

Die Qualitätsanforderungen und insbesondere der Auswertungsprozess dieser Arbeit sollen in ein komplexeres Empfehlungssystem integriert werden, das in erster Linie auf den Abgleich von Nutzerintentionen und annotierten Capabilitys setzt und weitere Kriterien berücksichtigt, vgl. [Rad+12]. Außer den Erfüllungsgraden der Qualitätsanforderungen werden die Verwendungshäufigkeiten, Heuristiken aus der gemeinsamen Verwendung von Kompositionsfragmenten innerhalb verschiedener Anwendungskontexte und kollaboratives Filtern genutzt. Ziel dabei ist es, aufgrund vieler Einflussparameter ein umfassendes Empfehlungssystem für Kompositionsfragmente in Web-Mashups zu entwickeln. Eine Herausforderung hierbei ist das Konfigurieren und Wichten der Einzelbewertungen, die zu einem komplexen Auswertungsergebnis führen, das schließlich für die verbesserte Empfehlung genutzt wird.

Die Forschungsarbeit um [MM14] beschäftigt sich mit dem Etablieren, Konfigurieren und Betreiben von Multi-Device-Mashups. Dabei werden komposite Web-Mashups als verteilte Anwendungen auf mehreren Geräten ausgeführt. Grundlage hierfür ist die – auch in dieser Arbeit genutzte – verteilte Laufzeitumgebung. Die regelbasierten Qualitätsanforderungen können als Ergebnis auch für die Verteilung von Anwendungsbestandteilen in solchen Multi-Device-Mashups sinnvoll genutzt werden. Das Zusammenspiel zwischen nutzergenerierten und gerätespezifischen Anforderungen könnte vor allem eine Weiterführung der Ergebnisse für das Szenario ③ mit hinterlegten Qualitätsanforderungen im Kontextmodell interessant machen. Mit dem in dieser Arbeit präsentierten Konzept für Qualitätsanforderungen können damit automatisiert Verteilungsoptionen in einer Umgebung mit verteilten Geräten in einem Anwendungskontext berechnet, vorgeschlagen und angewendet werden. Hier bietet sich die Erweiterung des Modells für Qualitätseigenschaften um solche für Geräte und mobile Plattformen an.

Sowohl das Einbeziehen standardisierter Komponentenmodelle als auch ein alternatives Entwicklungsparadigma für Web-Mashups wurden im Forschungsprojekt OMELETTE behandelt. Die vom W3C vorgeschlagenen Packaged Web Apps wurden dort auf eine Erweiterung zur Inter-Widget-Kommunikation untersucht [Chu+13]. Allgemein könnte ein *Importfilter für solche und andere Komponentenformate* etabliert werden. Hierbei ist im Kontext dieser Arbeit besonders das Importieren von Qualitätseigenschaften bzw. das Bereitstellen von Sensorik interessant. Außerdem wurde in diesem Forschungskontext das subtraktive Spezifizieren der Anwendungskommunikation untersucht. Alle potenziell kompatiblen Kommunikationsschnittstellen sind in diesem Fall miteinander standardmäßig verbunden. Unerwünschte Kommunikationsbeziehungen müssen entfernt werden. Dies ist im Kontext der qualitätsgetriebenen Empfehlung von Kompositionsfragmenten ein weiterer interessanter Untersuchungsgegenstand.

Obwohl [Tie15] die Verortung von Qualitätseigenschaften in Aufgabenmodellen und deren Abbildung auf Kompositionsvorschläge grundsätzlich behandelt, könnte eine umfassende Betrachtung der *Importstrategien für Drittmodelle*, insbesondere aus dem Bereich der Geschäftsprozesse, den Zugang für den Anwendungstyp Web-Mashup weiter steigern. Hierbei kommt es vor allem auf das gemeinsame Verwenden abgesprochener bzw. standardisierter Datenformate an. Die konsequente Modellierung der Konzepte dieser Arbeit mittels semantischer Formate zur Wissensmodellierung (RDF und OWL) liefert einen Beitrag dazu.

Die umfassende Nutzung von Qualitätsanforderungen und die Bemühungen der Generierung solcher Anforderungen aus verschiedenen Quellen legt die Frage nahe, wie mit der zeitlichen Gültigkeit bzw. dem *Altern von Qualitätsanforderungen* umgegangen werden soll. Diese Arbeit ermöglicht über die Konzepte der Nutzerprofile und der damit verbundenen Speichermöglichkeit für Favoriten deren dauerhaftes Vorhalten bzw. ihr wiederkehrendes Wirksamwerden. In künftigen Forschungsarbeiten könnte darauf aufbauend untersucht werden, nach welchen Kriterien sich Anforderungen abschwächen oder gar verschwinden sollen. Dazu zählt auch das erneute Bestätigen von Anforderungen, die eine bestimmte Alterungsfrist überschreiten, bzw. eine Erinnerungsfunktion.

Ausbaupotenzial hat zudem die Betrachtung der Konfliktprävention und -behandlung beim zeitnahen Aufeinandertreffen von Aktionen, die durch die Auswertung von Qualitätsanforderungen ausgelöst wurden. In dieser Arbeit wurden die verschiedenen Arten von Konfliktsituationen grundlegend behandelt und auch einige Maßnahmen zu Prävention genannt, wie beispielsweise das Einführen pessimistischer Nebenläufigkeitsbehandlung durch das Verbieten von neuen Aktionen mit Konfliktpotenzial, das sich aus der vorgestellten Konfliktmatrix ergibt. Anknüpfungspunkte für weitere Forschungsarbeiten gibt es in der domänenspezifischen Auswahl und dadurch der Einschränkung der Aktionsmenge und der interaktiven Konfiguration von Aktionen in Verbindung mit einer *Sandbox*-Ausführung von Kompositionsfragmenten.

Anhang A

Metamodelle und Schemata

Codeausschnitt A.1: Ausschnitt aus der XSD der erweiterten Metadaten für Komponentendeskriptoren in der SMCDL

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema targetNamespace="http://mmt.inf.tu-dresden.de/smcdl/1.15/metadata" elementFormDefault="
       qualified" attributeFormDefault="unqualified" version="1.15" xmlns:xs="http://www.w3.org/2001/
       XMLSchema" xmlns:meta="http://mmt.inf.tu-dresden.de/smcdl/1.15/metadata">
3      <xs:annotation>
4          <xs:documentation>
5              Meta information and quality properties for mashup components. Use this model as an extension
       to mashup component descriptors (SMCDL).
6          </xs:documentation>
7      </xs:annotation>
8
9      <xs:complexType name="Metadata">
10         <xs:annotation>
11             <xs:documentation>Collection of quality property values of type "Provided".</xs:documentation>
12         </xs:annotation>
13         <xs:sequence>
14             <xs:element ref="meta:qp" minOccurs="0" maxOccurs="unbounded"/>
15         </xs:sequence>
16     </xs:complexType>
17
18     <xs:complexType name="QualityProperty">
19         <xs:sequence>
20             <xs:element ref="meta:qp" minOccurs="0" maxOccurs="unbounded"/>
21         </xs:sequence>
22         <xs:attribute name="type" type="xs:string" use="required"/>
23         <xs:attribute name="concern" type="xs:string"/>
24         <xs:attribute name="value"/>
25     </xs:complexType>
26
27     <xs:element name="qp" type="meta:QualityProperty">
28         <xs:annotation>
29             <xs:documentation>
30               Quality property containing a type and optionally a concern. Either a value or further sub-
       elements must be provided in a composite fashion. Type values must be quality property types of
       class "Provided". See also the OWL-based quality properties reference model.
31             </xs:documentation>
32         </xs:annotation>
33     </xs:element>
34 </xs:schema>
```

Codeausschnitt A.2: Als TURTLE serialisiertes Metamodell für Qualitätseigenschaften mit dem Referenzmodell für Concerns

```
@prefix : <http://mmt.inf.tu-dresden.de/models/quality/property#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix dc: <http://purl.org/dc/elements/1.1/>.

<http://mmt.inf.tu-dresden.de/models/quality/property>
   rdf:type owl:Ontology;
   dc:title "Web Mashup Quality Properties"@en;
   dc:description "Quality properties metamodel for Web mashup composition fragments."@en.

### quality property metamodel top-level classes ###
:QualityPropertyCarryingEntity
   rdf:type owl:Class;
   rdfs:comment "Entity having quality properties, e. g., a Web mashup component or application."@en.

:QualityPropertyType
   rdf:type owl:Class;
   rdfs:comment "Type of a quality property."@en.

:QualityProperty
   rdf:type owl:Class;
   rdfs:comment "Quality property assigned to e. g. a Web mashup component having a type and a value."@en
        .

:DefaultQualityProperty
   rdf:type owl:Class;
   rdfs:subClassOf :QualityProperty;
   rdfs:comment "Assigns a default value to a quality property type such as 0, infinity, true, false,
        empty set or empty string. The default is itself of type quality property, since it is depending
        on the concern."@en.

:Concern
   rdf:type owl:Class;
   rdfs:comment "Concern defining a special aspect of a quality property."@en.

### types of value-assigning for quality properties ###
:Provided
   rdf:type owl:Class;
   rdfs:subClassOf :QualityPropertyType;
   rdfs:comment "Property type, which is provided statically, e. g. by the author."@en.

:Measurable
   rdf:type owl:Class;
   rdfs:subClassOf :QualityPropertyType;
   rdfs:comment "Property type with corresponding value determined at runtime."@en.

:Collectible
   rdf:type owl:Class;
   rdfs:subClassOf :QualityPropertyType;
   rdfs:comment "Property type with corresponding value collected over time using an aggregation rule."@
        en.

### properties for building relationships ###
:hasQualityProperty
   rdf:type owl:ObjectProperty;
   rdfs:domain [owl:unionOf(:QualityPropertyCarryingEntity :QualityProperty)];
   rdfs:range :QualityProperty;
   rdfs:comment "Assigns a property to a property carrying entity."@en.

:hasType
```

```
58    rdf:type owl:ObjectProperty;
59    rdfs:domain [owl:unionOf(:QualityProperty :FuzzySet)];
60    rdfs:range :QualityPropertyType;
61    rdfs:comment "Assigns a quality property type to a quality property or a fuzzy set aka. membership
          function."@en.
62
63  :hasConcern
64    rdf:type owl:ObjectProperty;
65    rdfs:domain :QualityProperty;
66    rdfs:range :Concern;
67    rdfs:comment "Assigns a concern to a quality property."@en.
68
69  :hasLiteralValue
70    rdf:type owl:DatatypeProperty;
71    rdfs:domain :QualityProperty;
72    rdfs:comment "Assigns a literal value to a quality property such as number or string."@en.
73
74  :hasCollectibleRawLiteralValue
75    rdf:type owl:DatatypeProperty;
76    rdfs:domain :QualityProperty;
77    rdfs:comment "Assigns a raw literal value to a collectible quality property such as number or string."
          @en.
78
79  ### fuzzy metamodel ###
80  :FuzzyTerm
81    rdf:type owl:Class;
82    rdfs:comment "Fuzzy term for assigning colloquial terms to quality property types."@en.
83
84  :FuzzySet
85    rdf:type owl:Class;
86    rdfs:comment "Membership function describing a fuzzy set for a specific property and colloquial term.
          Additionally, fuzzy sets may be user-specific."@en.
87
88  :hasFuzzyTerm
89    rdf:type owl:ObjectProperty;
90    rdfs:domain [owl:unionOf(:QualityPropertyType :FuzzySet)];
91    rdfs:range :FuzzyTerm;
92    rdfs:comment "Assigns a fuzzy term to a quality property type or a fuzzy set aka. membership function.
          "@en.
93
94  ### fuzzy sets: point chains as membership functions ###
95  :Point
96    rdf:type owl:Class;
97    rdfs:comment "Twodimensional points of property value and grade of membership to construct point
          chains."@en.
98
99  :onPropertyValue
100   rdf:type owl:DatatypeProperty;
101   rdfs:domain :Point;
102   rdfs:comment "Quality property value on which to assign a grade of membership within a membership
          function point."@en.
103
104 :hasGrade
105   rdf:type owl:DatatypeProperty;
106   rdfs:domain :Point;
107   rdfs:comment "Grade of membership for a fuzzy term at a specific property value."@en.
108
109 :PointChain
110   rdf:type owl:Class;
111   rdfs:subClassOf :FuzzySet;
112   rdfs:comment "Simple description form of a fuzzy set containing points."@en.
113
114 :hasPoint
```

```
115     rdf:type owl:ObjectProperty;
116     rdfs:domain :PointChain;
117     rdfs:range :Point;
118     rdfs:comment "Assigns a point to a point chain, spanning a membership function."@en.
119
120  ### concerns reference model ###
121  :API
122     rdf:type :Concern;
123     rdfs:comment "Concern pointing to the API of a black-box mashup component."@en.
124
125  :UI
126     rdf:type :Concern;
127     rdfs:comment "Concern pointing to the user interface of a mashup component."@en.
128
129  :Functionality
130     rdf:type :Concern;
131     rdfs:comment "Concern pointing to the features or capabilities of a mashup component."@en.
132
133  :Service
134     rdf:type :Concern;
135     rdfs:comment "Concern pointing to background Web services of a mashup component."@en.
136
137  :UIData
138     rdf:type :Concern;
139     rdfs:comment "Concern pointing to the data presented on the user interface of a mashup component."@en.
140
141  :BackgroundData
142     rdf:type :Concern;
143     rdfs:comment "Concern pointing to the data processed by the mashup component within the application
           logic of a mashup component."@en.
```

Codeausschnitt A.3: Als TURTLE serialisiertes Metamodell für Qualitätsanforderungen

```
1   @prefix : <http://mmt.inf.tu-dresden.de/models/quality/requirement#>.
2   @prefix qp:  <http://mmt.inf.tu-dresden.de/models/quality/property#>.
3   @prefix owl: <http://www.w3.org/2002/07/owl#>.
4   @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
5   @prefix dc: <http://purl.org/dc/elements/1.1/>.
6   @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
7
8   <http://mmt.inf.tu-dresden.de/models/quality/requirement>
9      rdf:type owl:Ontology;
10     owl:imports <http://mmt.inf.tu-dresden.de/models/quality/property>;
11     dc:title "Web Mashup Quality Requirements"@en;
12     dc:description "Quality requirements metamodel for developing and using Web mashup applications built
           from components."@en.
13
14  ### top-level ECA elements ###
15  :QualityRequirement
16     rdf:type owl:Class;
17     rdfs:comment "Quality requirement for composite Web mashups holding an ECA pattern."@en.
18
19  :Event
20     rdf:type owl:Class;
21     rdfs:comment "Constraints on the execution of requirements evaluation, e. g. time interval constraints
           ."@en.
22
23  :Condition
24     rdf:type owl:Class;
25     rdfs:comment "Abstract condition being the component in a composite pattern."@en.
26
27  :Action
28     rdf:type owl:Class;
```

```
29      rdfs:comment "Action to be performed in case of requirements condition violation."@en.
30
31  :hasEvent
32      rdf:type owl:ObjectProperty;
33      rdfs:domain :QualityRequirement;
34      rdfs:range :Event;
35      rdfs:comment "Assigns an event (evaluation configuration) to a quality requirement."@en.
36
37  :hasCondition
38      rdf:type owl:ObjectProperty;
39      rdfs:domain [owl:unionOf(:QualityRequirement :CompositeCondition)];
40      rdfs:range :Condition;
41      rdfs:comment "Assigns a condition to a quality requirement or to a composite condition."@en.
42
43  :hasAction
44      rdf:type owl:ObjectProperty;
45      rdfs:domain :QualityRequirement;
46      rdfs:range :Action;
47      rdfs:comment "Assigns an action and possibly a chain of further actions to a quality requirement."@en.
48
49  ### special conditions ###
50  :AtomicCondition
51      rdf:type owl:Class;
52      rdfs:subClassOf :Condition;
53      rdfs:comment "Leaf of a condition tree having an actual condition using an operator."@en.
54
55  :CompositeCondition
56      rdf:type owl:Class;
57      rdfs:subClassOf :Condition;
58      rdfs:comment "Node of a condition tree having two descendants with a logical connective."@en.
59
60  :FuzzyCondition
61      rdf:type owl:Class;
62      rdfs:subClassOf :AtomicCondition;
63      rdfs:comment "Condition having a fuzzy term instead of a discrete compare value."@en.
64
65  :DiscreteCondition
66      rdf:type owl:Class;
67      rdfs:subClassOf :AtomicCondition;
68      rdfs:comment "Condition having a discrete compare value and an appropriate operator."@en.
69
70  :hasCompareValue
71      rdf:type owl:DatatypeProperty;
72      rdfs:domain :DiscreteCondition;
73      rdfs:comment "Assigns a compare value to a discrete condition."@en.
74
75  :hasWeight
76      rdf:type owl:DatatypeProperty;
77      rdfs:domain :Condition;
78      rdfs:comment "Weight to be assigned to a condition, especially useful within a complex hierarchy of
            conditions."@en.
79
80  ### operators and connectives ###
81  :Operator
82      rdf:type owl:Class;
83      rdfs:comment "Compare operator specifying the required relation between feature value (value of
            quality property) and a fuzzy term or a compare value."@en.
84
85  :hasOperator
86      rdf:type owl:ObjectProperty;
87      rdfs:domain :AtomicCondition;
88      rdfs:range :Operator;
89      rdfs:comment "Assigns an operator to an atomic condition. This is especially useful for discrete
```

```
        conditions, but also applicable to fuzzy conditions in certain situations."@en.

:Contains rdf:type :Operator.
:EqualTo rdf:type :Operator.
:GreaterThan rdf:type :Operator.
:GreaterThanOrEqualTo rdf:type :Operator.
:LessThan rdf:type :Operator.
:LessThanOrEqualTo rdf:type :Operator.
:NotEqualTo rdf:type :Operator.

:Connective rdf:type owl:Class;
   rdfs:comment "Logical connective to connect the children of multiple composite conditions."@en.

:hasConnective
   rdf:type owl:ObjectProperty;
   rdfs:domain :CompositeCondition;
   rdfs:range :Connective;
   rdfs:comment "Assigns a logical connective such as AND or OR to a composite condition."@en.

:And rdf:type :Connective.
:Or rdf:type :Connective.

### relations to property model ###
:onFeature
   rdf:type owl:DatatypeProperty;
   rdfs:domain :AtomicCondition;
   rdfs:comment "Assigns a feature (quality property type) to a selector corresponding to a requirements
        condition. The feature is addressed using a SPARQL property path string."@en.

:onConcern
   rdf:type owl:ObjectProperty;
   rdfs:domain :AtomicCondition;
   rdfs:range qp:Concern;
   rdfs:comment "Assigns a concern to an atomic condition."@en.

:onTerm
   rdf:type owl:ObjectProperty;
   rdfs:domain :FuzzyCondition;
   rdfs:range qp:FuzzyTerm;
   rdfs:comment "Assigns a fuzzy term to a fuzzy condition."@en.
```

Codeausschnitt A.4: Als TURTLE serialisiertes Modell zur Aggregation von Werten für Qualitätseigenschaften auf Anwendungsebene mit Zuordnung von Funktionen auf Datentypen

```
@prefix : <http://mmt.inf.tu-dresden.de/models/quality/property-aggregation#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix dc: <http://purl.org/dc/elements/1.1/>.
@prefix xml: <http://www.w3.org/XML/1998/namespace>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

<http://mmt.inf.tu-dresden.de/models/quality/property-aggregation>
  rdf:type owl:Ontology;
  owl:imports <http://mmt.inf.tu-dresden.de/models/quality/property>;
  dc:title "Web Mashup Quality Properties Aggregation"@en;
  dc:description "Application-level aggregation functions metamodel for Web mashup quality properties."@
       en.

### aggregation data types ###
:AggregationDatatype
  rdf:type owl:Class.
```

```
18
19  :Boolean
20    rdf:type owl:Class;
21    rdfs:subClassOf :AggregationDatatype.
22
23  :Number
24    rdf:type owl:Class;
25    rdfs:subClassOf :AggregationDatatype.
26
27  :String
28    rdf:type owl:Class;
29    rdfs:subClassOf :AggregationDatatype.
30
31  ### types of aggregation functions ###
32  :AggregationType
33    rdf:type owl:Class.
34
35  :SelectOneValue
36    rdf:type owl:Class;
37    rdfs:subClassOf :AggregationType.
38
39  :Maximum
40    rdf:type :SelectOneValue, :Number.
41
42  :Median
43    rdf:type :SelectOneValue, :Number.
44
45  :Minimum
46    rdf:type :SelectOneValue, :Number.
47
48  :Mode
49    rdf:type :SelectOneValue, :Number, :Boolean.
50
51  :CalculateOneValue
52    rdf:type owl:Class;
53    rdfs:subClassOf :AggregationType.
54
55  :Average
56    rdf:type :Number, :CalculateOneValue.
57
58  :Sum rdf:type :Number, :CalculateOneValue.
59
60  :Count
61    rdf:type :String, :Number, :CalculateOneValue.
62
63  :ConcatAndGroup
64    rdf:type :Boolean, :Number, :CalculateOneValue.
65
66  :SelectAllValues
67    rdf:type owl:Class;
68    rdfs:subClassOf :AggregationType.
69
70  :Concatenation
71    rdf:type :String, :SelectAllValues.
```

Anhang B

Referenzmodelle

Die nachfolgend angeführten Qualitätseigenschaften mit Zuordnung von *Concerns* ergänzen die exemplarisch vorgestellten Einträge aus dem Referenzmodell in Unterabschnitt 4.6.1.

QE Angemessenheit der Komponenten (Component Suitability)

Wie gut sind die integrierten Komponenten zur Verwendung innerhalb einer bestimmten Anwendung aus Sicht dieser Anwendung geeignet?

- **Ermittlung:** gesammelt, **Datentyp:** Float, **Vorgabewert:** 0
- **Concerns:** API, UI-Daten, Hintergrunddaten, Funktionalität, Hintergrunddienste, UI
- **Terme:** niedrig, mittel, möglichst hoch, hoch
- **Anwendungsmetrik:** Es werden die Bewertungen der Nutzer und Entwickler gesammelt, die angeben, wie passend jeder Concern einer Komponente für das Kompositionsmodell der Anwendung ist. Die Angemessenheit des Komponenten-API und der Daten im Hintergrund sollten nur von Entwicklern bzw. Experten bewertet werden, da Nutzer dies nicht ohne Weiteres bewerten können. Zusätzlich kann der Mittelwert der Bewertungen für alle Concerns als Wert für die Angemessenheit der gesamten Komponente angegeben werden.

QE Anpassbarkeit (Adaptability)

Wie gut lässt sich die Funktionalität oder die Benutzeroberfläche ändern bzw. wie gut passen sich angezeigte Daten an wechselnde Kontexte an?

- **Ermittlung:** gesammelt, **Datentyp:** Float (Five-Star-Rating), **Vorgabewert:** 0
- **Concerns:** UI-Daten, Funktionalität, UI
- **Terme:** niedrig, mittel, möglichst hoch, hoch
- **Komponenten- und Anwendungsmetrik:** Es werden jeweils die diesbezüglichen Bewertungen der Nutzer und Entwickler gesammelt und deren Durchschnitt als Wert angegeben.

QE Datenverkehr (Data Traffic)

Welcher Datenverkehr kommt für die Komponente oder Anwendung durchschnittlich auf? Dies ist beispielsweise für schlecht angebundene Netzwerke oder bei volumenbasierter Abrechnung wichtig.

- **Ermittlung:** dynamisch und gesammelt, **Dimension:** Datenmenge pro Zeit (s^{-1})
- **Concerns:** Hintergrunddaten, **Vorgabewert:** ∞
- **Terme:** hoch, mittel, möglichst niedrig, niedrig
- **Komponenten- und Anwendungsmetrik:** Gemessen wird das verbrauchte Datenvolumen jeweils innerhalb eines bestimmten Intervalls der Nutzungszeit t. Es wird dann der Durchschnittswert aus den gemessenen Werten angegeben:

$$\text{Datenverkehr} = \frac{\text{Datenvolumen}}{t}$$

QE Erlernbarkeit (Learnability)

Wie leicht ist der Umgang mit der Anwendung oder Komponente zu erlernen bzw. zu verstehen? Ausschlaggebend neben dem angebotenen UI sind hier die API-Dokumentation und auch die Metadaten, die den Zweck der Komponente bzw. die zu erfüllenden Aufgaben und Funktionalitäten beschreiben.

- **Ermittlung:** gesammelt, **Datentyp:** Float (Five-Star-Rating), **Vorgabewert:** 0
- **Concerns:** UI, API (nur auf Komponentenebene)
- **Terme:** niedrig, mittel, möglichst hoch, hoch
- **Komponenten- und Anwendungsmetrik:** Die Erlernbarkeit wird von den Nutzern bzgl. des UI und von Entwicklern bzw. Experten bzgl. des Komponenten-API bewertet. Als Wert wird der Durchschnitt der Bewertungen angegeben.

QE Gebrauchstauglichkeit (Usability)

Wie gut lässt sich das API der Komponente bzw. das UI der Komponente oder Anwendung bedienen? Die Eigenschaft ist nicht in Einzelaspekte wie die Nutzerzufriedenheit unterteilt, da die Wertebelegung hier durch eine Nutzerbewertung zustande kommt und sich die Nutzer eine differenzierte Betrachtung nicht zumuten wollen.

- **Ermittlung:** gesammelt, **Datentyp:** Float (Five-Star-Rating), **Vorgabewert:** 0
- **Concerns:** API (für Komponenten), UI
- **Terme:** niedrig, mittel, möglichst hoch, hoch
- **Komponenten- und Anwendungsmetrik:** Die Benutzbarkeit wird von den Nutzern bzgl. des UI und von Entwicklern bzw. Experten bzgl. des Komponenten-API bewertet. Als Wert wird die durchschnittliche Bewertung angegeben.

QE Genauigkeit (Accuracy)

Wie exakt spiegeln die angezeigten Daten den zugrundeliegenden Sachverhalt wider? Sind die Daten korrekt bzw. wie hoch ist die Anzahl an Fehlern?

- **Ermittlung:** gesammelt, **Datentyp:** Float, **Vorgabewert:** 0
- **Concerns:** UI-Daten
- **Terme:** gering, mittel, möglichst hoch, hoch
- **Komponenten- und Anwendungsmetrik:** Da die Korrektheit von verarbeiteten Informationen nicht ohne Weiteres bei Komponenten und Anwendungen gemessen werden kann, werden hier Expertenbewertungen angegeben. Im Gegensatz dazu können Plausibilitätsprüfungen, die auf einer Grammatik oder auf einem Wertebereich basieren, über Qualitätsanforderungen an die Daten, die an den Kommunikationsschnittstellen der Mashup-Bestandteile anfallen, realisiert werden.

QE Gestaltung (Style)

Wie gestaltet sich die Benutzeroberfläche? Dazu zählen beispielsweise die dargestellte *Schrift* mit ihrer *Farbe*, *Größe* und *Art* sowie die *Hintergrundfarbe* der Komponente bzw. Anwendung. Die Eigenschaft zerfällt somit in mehrere der genannten Einzelangaben, die hier nicht separat modelliert sind und das Referenzmodell erweitern können. Anforderungen auf diese Gestaltungskenngrößen ermöglichen ein *Branding*, d. h. eine individuelle Anpassung, der gewählten Anwendungsbestandteile.

- **Ermittlung:** statisch, **Datentyp:** String, **Vorgabewert:** {}
- **Concerns:** UI
- **Komponentenmetrik:** Angabe der oben genannten Teilaspekte mit ihren jeweiligen Typen, meist als String.
- **Anwendungsmetrik:** Es werden alle Werte der enthaltenen Komponenten angegeben sowie, wenn möglich, auch Werte für global in der Anwendung verwendete Gestaltung.

QE Glaubwürdigkeit (Credibility)

Stammen die gezeigten Informationen aus einer vertrauenswürdigen Quelle? Hierbei ist als Rahmenbedingung zu beachten, dass eine Zertifizierungsinfrastruktur bereitgestellt wird oder genutzt wird.

- **Ermittlung:** statisch, **Dimension:** Boolean bzw. Integer, **Vorgabewert:** false bzw. 0
- **Concerns:** UI-Daten (indirekt: Hintergrunddienste und die von ihnen gelieferten Hintergrunddaten)
- **Terme:** vorhanden, nicht vorhanden, glaubwürdig, unglaubwürdig

- **Komponentenmetrik:** Angabe durch den Entwickler, ob der verwendete Hintergrunddienst ein Zertifikat liefert.
- **Anwendungsmetrik:** Bei der Anwendung wird die Anzahl der Komponenten angegeben, deren Hintergrunddienste Zertifikate liefern.

$$\text{Glaubwürdigkeit}_{\text{Anwendung}} = |\text{Komponenten mit Zertifikat}|$$

QE Interoperabilität (Interoperability)

Wie gut eignet sich das Komponenten-API, um in Anwendungen durch Verknüpfung mit anderen Komponenten eingebunden werden zu können?

- **Ermittlung:** statisch, **Datentyp:** Integer, **Vorgabewert:** 0
- **Concerns:** API, Hintergrunddaten entsprechend der Teileigenschaftsbezüge
- **Terme:** schlecht, mittelmäßig, möglichst gut, gut
- **Komponentenmetrik:** Es wird ein Wert angegeben, der die Anzahl der angebotenen verschiedenen Datenaustauschformate f, Protokolle p und Programmiersprachen l aggregiert:

  $$\text{Interoperabilität}_{\text{Komponenten-API}} = |f| + |p| + |l|$$

  Für diese Eigenschaft wird dementsprechend nicht für jeden Concern ein separater Wert angegeben, sondern nur ein Wert, der sich aus den Angaben der Teileigenschaften ergibt. Die Teileigenschaften werden alle statisch vom Entwickler angegeben. Entscheidend ist dabei ihre Anzahl, die während der Aggregation zu einer Gesamtsumme verrechnet wird.

QE Datenaustauschformate (Data Exchange Formats) ← Interoperabilität

Welche Datenaustauschformate, beispielsweise JSON, XML, Atom oder RSS, werden vom API der Komponenten angeboten?

- **Concerns:** API, Hintergrunddaten

QE Programmiersprachen (Programming Languages) ← Interoperabilität

Mit welchen Programmiersprachen bzw. welcher Programmiersprache (z. B. JavaScript) wurde die Funktionalität der Komponente programmiert?

- **Concerns:** API (indirekt auch Funktionalität)

QE Protokolle (Protocols) ← Interoperabilität

Welche Protokolle (z. B. die Architekturstile SOAP, Representational State Transfer (REST)) werden in der Komponente verwendet?

- **Concerns:** API

QE Konsistenz (Consistency)

Werden Design und Strukturen des UI sowie der Daten auf der Benutzeroberfläche in den in der Anwendung enthaltenen Komponenten konsistent verwendet? Läuft die Kommunikation über die Komponentenschnittstellen semantisch konsistent ab (Funktionalität)?

- **Ermittlung:** gesammelt, **Datentyp:** Float (Five-Star-Rating), **Vorgabewert:** 0
- **Concerns:** Funktionalität, UI, UI-Daten
- **Terme:** niedrig, mittelmäßig, möglichst hoch, hoch
- **Anwendungsmetrik:** Hier wird als aggregierter Wert der Durchschnitt von Nutzerbewertungen jeweils pro Concern angegeben.

QE Quelle (Source)

Aus welcher Quelle stammen verwendete Ressourcen, wie etwa der verwendete Hintergrunddienst, in der Komponente?

- **Ermittlung:** statisch, **Datentyp:** String, **Vorgabewert:** ""
- **Concerns:** Hintergrunddaten
- **Komponentenmetrik:** Angabe eines URI oder eines Anbieternamens durch den Entwickler.
- **Anwendungsmetrik:** Angabe aller Quellen der Komponenten.

QE Rechtzeitigkeit (Timeliness)

Führt die Anwendung oder Komponente Aufgaben innerhalb einer bestimmten – vom Nutzer bzw. Entwickler festgelegten – Frist aus? Diese Eigenschaft ist verwandt zur Antwortzeit, betrifft jedoch die Umsetzung der wahrgenommenen Reaktion, die von der tatsächlichen Antwortzeit unabhängig sein kann.

- **Ermittlung:** dynamisch und gesammelt, **Datentyp:** Float, **Vorgabewert:** 0
- **Concerns:** Hintergrunddienste, UI (indirekt: Funktionalität)
- **Terme:** gering, mittelmäßig, möglichst hoch, hoch
- **Komponentenmetrik:** Es wird bei jeder Messung überprüft, ob die angegebene Frist pro Komponente eingehalten wurde. Angegeben wird der Anteil der rechtzeitigen Reaktionen, d. h. der Antworten innerhalb einer festgelegten Frist:

$$\text{Rechtzeitigkeit}_{\text{Komponente}} = \frac{|\text{Rechtzeitige Reaktionen}|}{|\text{Messungen}|}$$

- **Anwendungsmetrik:** Es wird der Mittelwert aller Werte für die Rechtzeitigkeit der enthaltenen Komponenten jeweils für UI und Hintergrunddienste angegeben.

$$\text{Rechtzeitigkeit}_{\text{Anwendung}} = \text{avg}(\text{Rechtzeitigkeit}_{\text{Komponente}})$$

QE Frist (Deadline) ← Rechtzeitigkeit

Gibt die Frist an, innerhalb welcher Zeit der Hintergrunddienst bzw. das UI der Komponente typischerweise antwortet bzw. antworten soll.

- **Ermittlung:** statisch, **Dimension:** Zeit (s), **Vorgabewert:** 0
- **Concerns:** Hintergrunddienste, UI (indirekt: Funktionalität)
- **Terme:** hoch, mittelmäßig hoch, möglichst niedrig, niedrig
- **Komponentenmetrik:** Angabe einer Zeitspanne, die mit der Antwortzeit verglichen werden kann.

QE Sicherheit (Security)

Zu welchem Grad ist die Komponente oder Anwendung vor unbefugter Nutzung geschützt? Hierbei werden unterstützte Schutzmechanismen, wie das Anbieten von Transportverschlüsselung, gezählt.

- **Ermittlung:** statisch, **Datentyp:** Integer, **Vorgabewert:** 0
- **Concerns:** Funktionalität, Hintergrunddaten entsprechend der Teileigenschaften
- **Terme:** niedrig, mittel, möglichst hoch, hoch
- **Komponentenmetrik:** Es wird als Grad die Anzahl der verwendeten *Schutzmechanismen* angegeben. Ein Schutzmechanismus steht in diesem Fall für eine der Teileigenschaften.

$$\text{Sicherheit}_{\text{Komponente}} = |\text{verwendete Schutzmechanismen}_{\text{Komponente}}|$$

- **Anwendungsmetrik:** Es wird die Summe der Komponentenwerte angegeben.

Für die Sicherheit der Komponenten und Anwendungen wird nicht für jeden Concern ein extra Wert angegeben, da dies eine Eigenschaft ist, deren Wert sich in der angeführten Weise aus den Angaben der Teileigenschaften zusammensetzt.

QE Authentifizierung (Authentication) ← Sicherheit

Findet eine Authentifizierung statt?

- **Concerns:** Funktionalität

QE Autorisierung (Authorization) ← Sicherheit

Gibt es die Möglichkeit Nutzer zu autorisieren?

- **Concerns:** Funktionalität

QE Datenspeicherung (Data Storage) ← Sicherheit

Werden Daten während der Laufzeit zwischengespeichert?

- **Concerns:** Hintergrunddaten

QE Datenverschlüsselung (Data Encryption) ← Sicherheit

Wird eine Datenverschlüsselung, beispielsweise Transport Layer Security (TLS), im Backend verwendet?

- **Concerns:** Hintergrunddaten

QE Nachvollziehbarkeit (Traceability) ← Sicherheit

Ist die Nutzungshistorie einsehbar?

- **Concerns:** Funktionalität

QE Übertragungsrate (Transfer Rate)

Welche Datenübertragungsrate benötigt die Komponente oder Anwendung, um die vom Entwickler ermittelte volle Leistung erreichen zu können? Wie hoch muss die vorhandene Übertragungsrate mindestens sein, damit die Komponente oder Anwendung richtig ausgeführt werden kann? Diese Eigenschaft ist mit dem Datenverkehr verwandt. Dort wird allerdings keine typische Rate festgelegt, sondern die tatsächliche Rate gemessen.

- **Ermittlung:** statisch, **Dimension:** Datenmenge pro Zeit (s^{-1}), **Vorgabewert:** ∞
- **Concerns:** Hintergrunddaten
- **Terme:** hoch, mittel, möglichst niedrig, niedrig
- **Komponentenmetrik:** Angabe jeweils einer Datenübertragungsrate, die für die bestmögliche Leistung benötigt wird und einer Übertragungsrate, die minimal nötig ist, damit die Komponente ausgeführt werden kann. Die Werte werden vom Entwickler durch Tests bestimmt. Da hierbei nicht auf den Einfluss durch Geräten und Plattformen individuell eingegangen werden kann, werden Annahmen für zeitgemäße Systeme getroffen.
- **Anwendungsmetrik:** Für die Anwendung wird jeweils der höchste Wert aus allen Komponenten angegeben:

$$\text{Übertragungsrate}_{\text{Anwendung}} = \max(\text{Übertragungsrate}_i)$$

QE Vertrautheit (Familiarity)

Welche allgemein bekannten oder standardisierten Gestaltungselemente oder Bedienparadigmen werden, beispielsweise übliche Farbschemata bzw. Formen der UI-Elemente, verwendet?

- **Ermittlung:** statisch, **Datentyp:** String, **Vorgabewert:** »keine Standards« (`{}`)

- **Concerns:** UI
- **Komponentenmetrik:** Angabe verwendeter Standards, wie etwa verwendete Human Interface Guidelines und Bedienparadigmen (Drag & Drop).
- **Anwendungsmetrik:** Bei der Anwendung werden alle in den Komponenten verwendeten Standards und die Bedienparadigmen, die in der Anwendung für die Bedienung über mehrere Komponenten verwendet werden, angegeben.

QE Vollständigkeit (Completeness)

Sind alle Funktionen, Informationen, Gestaltungselemente, die zur Erfüllung der Aufgabe nötig sind, vorhanden?

- **Ermittlung:** gesammelt, **Datentyp:** Float, **Vorgabewert:** 0
- **Concerns:** Funktionalität, UI-Daten, Hintergrunddaten, UI, API bei der Komponentenebene
- **Terme:** unvollständig, halbwegs vollständig, möglichst vollständig, vollständig
- **Komponenten- und Anwendungsmetrik:** Ob eine Anwendung oder Komponente alle notwendigen Funktionen und Informationen bereithält, ist abhängig von den Anforderungen der Nutzer in Bezug auf die Anwendungen und die der Anwendungsentwickler und Nutzer in Bezug auf die Komponenten. Die Vollständigkeit des Komponenten-API und der Daten im Hintergrund von Komponenten und Anwendungen wird von Experten bewertet. Es wird ein Anteil zwischen 0 und 1 angegeben.

QE Zugänglichkeit (Accessibility)

Wie gut ist die Benutzerschnittstelle mit ihren Bedienparadigmen und ihrer Gestaltung für verschiedene Nutzertypen zugänglich? Dies impliziert auch die Zugänglichkeit der gezeigten Informationen. Sind wichtige Funktionen immer sichtbar und verfügbar?

- **Ermittlung:** gesammelt, **Datentyp:** Float, **Vorgabewert:** 0
- **Concerns:** UI (indirekt: UI-Daten)
- **Terme:** schlecht, mittelmäßig, möglichst gut, gut
- **Komponenten- und Anwendungsmetrik:** Die Zugänglichkeit wird von den Nutzern bewertet. Es wird der Durchschnitt der gesammelten Bewertungen angegeben.

QE Zuverlässigkeit (Reliability)

Mit welcher Wahrscheinlichkeit kann die Komponente oder Anwendung fehlerfrei ausgeführt werden?

- **Ermittlung:** gesammelt, **Datentyp:** Float, **Vorgabewert:** 0

- **Concerns:** Funktionalität, Hintergrunddienste, UI (indirekt: UI-Daten, Hintergrunddaten) entsprechend der Concerns der Teileigenschaften
- **Komponentenmetrik:** Die Eigenschaft gibt an, wie zuverlässig die Komponente läuft. Dafür wird der Durchschnitt aus den Werten für die Robustheit r und die Verfügbarkeit der Funktionalität a_f und des Hintergrunddienstes a_s der Komponente berechnet:

$$\text{Zuverlässigkeit}_{\text{Komponente}} = \frac{a_f + a_s + r}{3}$$

- **Terme:** niedrig, mittel, möglichst hoch, hoch, zuverlässig, unzuverlässig
- **Anwendungsmetrik:** Für die Anwendung wird der Durchschnitt über die Werte der Komponenten angegeben.

$$\text{Zuverlässigkeit}_{\text{Anwendung}} = \text{avg}(\text{Zuverlässigkeiten}_{\text{Komponente}})$$

  Bei der Zuverlässigkeit wird, wie bei der Interoperabilität und der Sicherheit, nicht jeweils ein Wert für jeden Concern angegeben, da sie als übergeordnete Eigenschaft nur einen zusammenfassenden Wert aus den Werten der Teileigenschaften repräsentiert. Als Aggregation bietet sich alternativ auch die Minimumsfunktion an.

QE Robustheit (Robustness) ← Zuverlässigkeit

Wie resistent ist die Anwendung oder Komponente gegen Fehler bzgl. der Funktionalität? Für die Aggregation zu Nutzung innerhalb der Zuverlässigkeit wird der Robustheitswert noch normiert.

- **Ermittlung:** dynamisch und gesammelt, **Datentyp:** Float, **Vorgabewert:** 0
- **Concerns:** Funktionalität, Hintergrunddienste, UI (indirekt: UI-Daten, Hintergrunddaten)
- **Terme:** niedrig, mittel, möglichst hoch, hoch, robust
- **Komponenten- und Anwendungsmetrik:** Nutzungszeit t pro auftretender Ausnahme:

$$\text{Robustheit} = \frac{t}{|\text{Ausnahmen}|}$$

QE Verfügbarkeit (Availability) ← Zuverlässigkeit

Mit welcher Wahrscheinlichkeit sind Funktionen oder der Hintergrunddienst erreichbar?

- **Ermittlung:** dynamisch und gesammelt, **Datentyp:** Float, **Vorgabewert:** 0
- **Concerns:** Funktionalität, Hintergrunddienste, UI (indirekt: UI-Daten, Hintergrunddaten)
- **Terme:** niedrig, mittel, möglichst hoch, hoch, hochverfügbar

- **Komponentenmetrik:** Anteil der erfolgreichen Verbindungen c_s an der Anzahl der Verbindungsversuche c:
$$\text{Verfügbarkeit}_{\text{Service}} = \frac{c_s}{c}$$
Alternativ können auch die verbindungslosen Zeiten pro Gesamtnutzungszeit angegeben werden.
- **Anwendungsmetrik:** Für die Verfügbarkeit der Services wird ein Durchschnittswert aus allen Werten der Komponenten angegeben.
$$\text{Verfügbarkeit}_{\text{Anwendungsservices}} = \text{avg}(\text{Verfügbarkeit}_{\text{Komponentenservice}})$$

Die nachfolgend angeführten Qualitätseigenschaften ohne die Zuordnung bestimmter Concerns ergänzen die exemplarisch vorgestellten Einträge aus dem Referenzmodell in Unterabschnitt 4.6.2.

QE Ansprechpartner (Maintainer)

Gibt den Namen und weitere Informationen, wie Anschrift, Telefonnummer, E-Mail-Adresse, Titel und Web-URI, zu der Person an, die verantwortlich für beispielsweise spätere Änderungen an der Komponente oder Anwendung ist, vgl. Entwickler.

- **Ermittlung:** statisch, **Datentyp:** String, **Vorgabewert:** jeweils `""`

QE Arbeitsspeicher (RAM)

Wie viel Speicher belegt die Komponente oder Anwendung im Arbeitsspeicher des Client-Rechners?

- **Ermittlung:** statisch, **Datentyp:** Datenmenge (bit), **Vorgabewert:** ∞
- **Terme:** viel, mittelmäßig viel, möglichst wenig, wenig
- **Komponenten- und Anwendungsmetrik:** Der belegte Speicherplatz im Arbeitsspeicher (RAM) ist vor allem in webbasierten Laufzeitumgebungen schwer messbar. Daher wird hier ein unter Testbedingungen vom Entwickler ermittelter Wert angegeben.

QE Bezeichner (Identifier, ID)

Gibt den Bezeichner der Komponente bzw. Anwendung zur eindeutigen Identifizierung an.

- **Ermittlung:** statisch, **Datentyp:** String, **Vorgabewert:** Angabe ist obligatorisch

QE Dimension

Gibt entweder eine *feste Höhe und Breite* an oder eine *minimale Höhe und Breite* und bzw. oder *die maximale Höhe und Breite* der Komponente bzw. Anwendung.

- **Ermittlung:** statisch, **Datentyp:** Pixel, **Vorgabewert:** 0 bzw. ∞
- **Terme:** gering, mittel, möglichst groß, groß (für maximale Höhe und Breite) und groß, mittel, möglichst gering, gering (für die minimale Höhe und Breite)

QE Dokumentation (Documentation)

Eine kurze Beschreibung der Komponente oder Anwendung.

- **Ermittlung:** statisch, **Datentyp:** String, **Vorgabewert:** ""
- **Terme:** kurz, lang, ausführlich (nach der Länge des Wertes)

QE Entwickler (Author)

Gibt den Namen des Komponenten- bzw. Anwendungsentwicklers an. Weiterhin ist die Angabe der Anschrift, der E-Mail-Adresse, des Titels, der Telefon- und Faxnummer sowie der Web-URI möglich.

- **Ermittlung:** statisch, **Datentyp:** jeweils String, **Vorgabewert:** jeweils ""

QE Erstellungsdatum (Creation Date)

Datum, an dem die Komponente oder Anwendung erstellt wurde. Für solche Metadaten bietet es sich an, vorhandene Domänenmodelle zu nutzen, wie etwa Dublin Core.

- **Ermittlung:** statisch, **Datentyp:** Zeitpunkt (Datum)
- **Vorgabewert:** 1. Januar 1970, 0 Uhr, **Terme:** alt, möglichst neu, neu

QE Gerät (Device)

Geräteprofil, auf dem die Anwendung oder Komponente ausgeführt werden kann. Hierbei wird ein Modell zur Beschreibung der Gerätefähigkeiten zu Hilfe genommen, siehe beispielsweise [MM14] für das Verteilen kompositer Anwendungen auf mehrere Geräte.

- **Ermittlung:** statisch, **Datentyp:** String, **Vorgabewert:** {}

QE Kategorie (Category)

Gibt ein Themengebiet an, das grob die Art einer Komponente oder Anwendung beschreibt, beispielsweise »Tourismus«, »Finanzen« oder »Unterhaltung«.

- **Ermittlung:** statisch, **Datentyp:** String, **Vorgabewert:** ""

QE Komponentenanzahl (Number of Components)

Gibt an, wie viele Komponenten sich in einer Anwendung befinden. Diese Eigenschaft existiert selbstverständlich nur auf Anwendungsebene und wird durch die Aggregation des Wertes 1 pro Komponente gebildet.

- **Ermittlung:** statisch, **Datentyp:** Integer, **Vorgabewert:** 0
- **Terme:** hoch, mittel, möglichst gering, gering

QE Lizenz (License)

Gibt an, unter welcher Lizenz die Anwendung oder Komponente im Kontext einer Laufzeitumgebung für Mashups verwendet werden kann. Dabei handelt es sich entweder um eine kommerzielle oder eine freie Softwarelizenz. Für die Lizenz wird eine Versionsnummer angegeben.

- **Ermittlung:** statisch, **Datentyp:** String, **Vorgabewert:** »keine Lizenz« (»no license«)
- **Terme:** GPL, MIT, CC, Public Domain, proprietär (ggf. mit entsprechender Versionsnummer)

QE Name

Gibt den Namen der Anwendung oder Komponente an.

- **Ermittlung:** statisch, **Datentyp:** String, **Vorgabewert:** ""

QE Schlüsselwort (Keyword)

Schlüsselwort, das die Anwendung oder Komponente passend beschreibt.

- **Ermittlung:** statisch, **Datentyp:** String, **Vorgabewert:** ""

QE Screenshot

Gibt den URI zu einem Screenshot der Komponente oder Anwendung an.

- **Ermittlung:** statisch, **Datentyp:** URI (String), **Vorgabewert:** ""

QE Sensor

Gibt an, welcher Sensor von der Komponente oder Anwendung verwendet wird. Sensoren werden in der Laufzeitumgebung, vor allem auf mobilen Geräten zur Bestimmung volatiler Messwerte genutzt.

- **Ermittlung:** statisch, **Datentyp:** String, **Vorgabewert:** ""

QE Signatur (Signature)

Gibt die digitale Signatur der Anwendung bzw. Komponente an. Die Signatur ordnet das Kompositionsfragment einem bestimmten Anbieter zu. Sie bestätigt, dass der Inhalt nicht zwischendurch verändert wurde.

- **Ermittlung:** statisch, **Datentyp:** String, **Vorgabewert:** »keine Signatur« (»no signature«)
- **Terme:** vorhanden, nicht vorhanden

QE Speicherplatz (Memory)

Wie viel Speicher belegt die Komponente oder Anwendung auf dem Rechner typischerweise während ihrer Ausführung in der Mashup-Laufzeitumgebung?

- **Ermittlung:** statisch, **Datentyp:** Datenmenge (bit), **Vorgabewert:** ∞
- **Terme:** hoch, mittel, möglichst niedrig, niedrig
- **Komponentenmetrik:** Statische Angabe des benötigten Speicherbedarfs für eine bestimmte Komponente. Dieser wird durch Tests ermittelt. Auf die Abhängigkeiten zur Hardware und zum Betriebssystem sowie weiteren Größen wie der Netzwerkanbindung wird hier keine Rücksicht genommen.
- **Anwendungsmetrik:** Für die Anwendung wird für den belegten Speicherplatz die Summe der jeweiligen Komponentenwerte angegeben.

QE Sprache (Language)

Gibt die Sprache an, die das UI der Komponente bzw. der Anwendung unterstützt.

- **Ermittlung:** statisch, **Datentyp:** String (Sprachkürzel nach ISO 639), **Vorgabewert:** `""`

QE UI

Gibt die Art der Komponente – UI- oder Service – an. Eine UI-Komponente bietet eine Benutzerschnittstelle an, wohingegen eine Service-Komponente nicht sichtbar ist und in der Anwendung nur im Hintergrund agiert. Diese Eigenschaft gilt selbstverständlich nur auf Komponentenebene. Sie ist wichtig, um zu entscheiden, ob die Komponente auch für die Ausführung in einem serverseitigen Kontext in Frage kommt. Der Wert `true` steht für eine UI-Komponente, `false` für eine Service-Komponente.

- **Ermittlung:** statisch, **Datentyp:** Boolean, **Vorgabewert:** `false`
- **Terme:** ist UI-Komponente, ist keine UI-Komponente

QE URI

Gibt die Adresse an, unter der die Komponente oder Anwendung als Ressource im Web zu finden ist.

- **Ermittlung:** statisch, **Datentyp:** URI (String), **Vorgabewert:** `""`

QE Version

Gibt die Versionsnummer der Komponente oder Anwendung an.

- **Ermittlung:** statisch, **Datentyp:** Float, **Vorgabewert:** `"0.1.0"`
- **Terme:** niedrig, mittel, möglichst hoch, hoch

Codeausschnitt B.1: Ausschnitt aus dem Referenzmodell für Ereignisse zum Auslösen der Auswertung von Qualitätsanforderungen

```
:AtPointInTime
   a owl:Class;
   rdfs:subClassOf :Event;
   rdfs:comment "Evaluate requirement once at a specific point in time or immediately."@en.
   # :hasTargetTime ... calender widget or text field

:hasTargetTime
   a owl:DatatypeProperty;
   rdfs:domain :AtPointInTime;
   rdfs:comment "Requirements evaluation execution time."@en.

:Interval
   a owl:Class;
   rdfs:subClassOf :Event;
   rdfs:comment "Evaluate requirement repeatedly after a given time interval. Stops after a given amount
        of executions or never if amount is missing."@en.

:hasInterval # UI: time slider 5 s to 10 min
   a owl:DatatypeProperty;
   rdfs:domain :Interval;
   rdfs:comment "Time interval in milliseconds."@en.

:hasAmount
   a owl:DatatypeProperty;
   rdfs:domain :Interval;
   rdfs:comment "Integer amount of executions."@en.

:OnCommunicationEvent
   a owl:Class;
   rdfs:subClassOf :Event;
   rdfs:comment "Evaluate requirement on firing a specified communication event."@en.

:hasCommunicationEvent # drop down select menu for communication events in current app, consider using
     proper semantic concept from smcdl model
   a owl:DatatypeProperty;
   rdfs:domain :OnCommunicationEvent;
   rdfs:comment "Communication event name to listen to."@en.

:OnApplicationModelChange
   a owl:Class;
   rdfs:subClassOf :Event;
   rdfs:comment "Evaluate requirement on any development action within the application."@en.
   # examples: adding components, communication model changes, consider narrowing down modification types
        with parameter

:NearGeoLocation
   a owl:Class;
   rdfs:subClassOf :Event;
   rdfs:comment "Experimental: Evaluate requirement when approaching a given location with client device.
        "@en.

:hasLocation # map marker or coordinates input
   a owl:DatatypeProperty;
   rdfs:domain :NearGeoLocation;
   rdfs:comment "Geo location coordinates to monitor approach to. Specify coordinates format when
        required."@en.
```

Codeausschnitt B.2: Ausschnitt aus dem Referenzmodell für Aktionen zur Adaption in Abhängigkeit des Ausganges der Auswertung von Qualitätsanforderungen

```
### feedback actions ###
:ShowPopup
   a owl:Class;
   rdfs:subClassOf :Action;
   rdfs:comment "Show a text message."@en.
   # :hasMessage ... consider providing wildcards for component identifier

:hasMessage # consider providing wildcards for component identifier
   a owl:DatatypeProperty;
   rdfs:domain :ShowPopup;
   rdfs:comment "Message string to show."@en.

:RequestUserFeedback
   a owl:Class;
   rdfs:subClassOf :Action;
   rdfs:comment "Request user feedback by showing a dialog for text input or scale rating."@en.

:RequestExpertReview # use collaboration infrastructure, cf. GB
   a owl:Class;
   rdfs:subClassOf :Action;
   rdfs:comment "Experimental: Share the application with an expert."@en.

### application model modification actions ###
:ModifyComponent # @see :hasObsoleteComponent
   a owl:Class;
   rdfs:subClassOf :Action;
   rdfs:comment "Add, remove or replace a mashup component."@en.

:hasObsoleteComponent # UI: select from app model
   a owl:DatatypeProperty;
   rdfs:domain :ModifyComponent;
   rdfs:comment "Instance identifier of the mashup component to be removed or replaced."@en.

:hasNewComponent # UI: select from CoRe
   a owl:DatatypeProperty;
   rdfs:domain :ModifyComponent;
   rdfs:comment "Identifier of the mashup component to be added freshly or as a replacement."@en.

:ChangeView
   a owl:Class;
   rdfs:subClassOf :Action;
   rdfs:comment "Perform view transition to a referenced target layout."@en.

:hasTargetLayout # cf. layout model for relations between layout and view
   a owl:DatatypeProperty;
   rdfs:domain :ChangeView;
   rdfs:comment "Target layout to perform the view transition to."@en.

:ModifyCommunication
   a owl:Class;
   rdfs:subClassOf :Action;
   rdfs:comment "Experimental: Change application communication."@en.

:ModifyLayout
   a owl:Class;
   rdfs:subClassOf :Action;
   rdfs:comment "Experimental: Change layout."@en.

:ModifyDistribution # cf. distributed application execution concepts of Oliver Mroß
   a owl:Class;
   rdfs:subClassOf :Action;
```

```
62    rdfs:comment "Experimental: Change application distribution over devices and runtime environments."@en
          .
63
64  ### component implementation and binding actions ###
65  :SendApplicationMessage
66     a owl:Class;
67     rdfs:subClassOf :Action;
68     rdfs:comment "Send a message by calling the target operation to trigger component-internal adaptation.
           "@en.
69
70  :hasCommunicationOperation # operation to call as well as arguments
71     a owl:DatatypeProperty;
72     rdfs:domain :SendApplicationMessage;
73     rdfs:comment "Target operation comprising a component instance identifier as well as parameters."@en.
74
75  :ReplaceBinding
76     a owl:Class;
77     rdfs:subClassOf :Action;
78     rdfs:comment "Experimental: Replace the component binding aka. component implementation."@en.
79
80  ### runtime environment actions ###
81  :RecommendComponentReplacement
82     a owl:Class;
83     rdfs:subClassOf :Action;
84     rdfs:comment "Recommend a new component to replace an existing one."@en.
85
86  :hasComponentToBeReplaced # UI: select from app model
87     a owl:DatatypeProperty;
88     rdfs:domain :RecommendComponentReplacement;
89     rdfs:comment "Instance identifier of the mashup component to be removed or replaced."@en.
90
91  :MigrateComponent
92     a owl:Class;
93     rdfs:subClassOf :Action;
94     rdfs:comment "Experimental: Migrate a component to another device or platform."@en.
95
96  :hasComponentToBeMigrated
97     a owl:DatatypeProperty;
98     rdfs:domain :MigrateComponent;
99     rdfs:comment "Instance identifier of the mashup component to be migrated."@en.
100
101 :hasTarget
102    a owl:DatatypeProperty;
103    rdfs:domain :MigrateComponent;
104    rdfs:comment "Target platform identifier, e. g. server or other client device."@en.
105
106 :Log
107    a owl:Class;
108    rdfs:subClassOf :Action;
109    rdfs:comment "Log a text message, which may be augmented with evaluation context data."@en.
110
111 :hasLogMessage
112    a owl:DatatypeProperty;
113    rdfs:domain :Log;
114    rdfs:comment "Log message string."@en.
115
116 :LoadApplication
117    a owl:Class;
118    rdfs:subClassOf :Action;
119    rdfs:comment "Load another application replacing the currently executes one."@en.
120
121 :hasApplication # UI: select from application repository
122    a owl:DatatypeProperty;
```

```
123     rdfs:domain :LoadApplication;
124     rdfs:comment "Application instance identifier."@en.
125 
126 :SwitchDevelopmentView
127     a owl:Class;
128     rdfs:subClassOf :Action;
129     rdfs:comment "Switch to one of the following: Live view, Capability view, Professional view."@en.
130 
131 :hasTargetView # UI: dropdown of available views
132     a owl:DatatypeProperty;
133     rdfs:domain :LoadApplication;
134     rdfs:comment "Target view to switch to."@en.
135 
136 :ActivateExplanationMode
137     a owl:Class;
138     rdfs:subClassOf :Action;
139     rdfs:comment "Activate explanation mode, providing highlighting of UI elements in a capability-driven
            workflow."@en.
140 
141 :ApplyQualityProfile
142     a owl:Class;
143     rdfs:subClassOf :Action;
144     rdfs:comment "Experimental: Apply a quality profile, i. e. a set of quality requirements. Enables meta
            -programming."@en.
145 
146 :hasQualityProfile
147     a owl:DatatypeProperty;
148     rdfs:domain :ApplyQualityProfile;
149     rdfs:comment "Quality profile aka. set of quality requirements to apply."@en.
```

Anhang C

Komponentenimplementierungen

Codeausschnitt C.1: Beispielimplementierung einer Mashup-Komponente für das Anzeigen des aktuellen Wetters an einem bestimmten Ort in JavaScript

```
"use strict";

/**
 * Displays the current weather for a given location.
 *
 * @see https://openweathermap.org/current
 */
class OpenWeatherMapWeather extends MashupComponent {
    constructor() {
        super();
        this.key = "162ab1fd8e8fe06f4c13eb935862da37";
        this.baseURI = "http://api.openweathermap.org/data/2.5/weather";
        this.iconURI = "http://openweathermap.org/img/w/";
        this.location = "";
        this.responseTime = 0;
        this.currentTemperature = 0;
    }

    init(context) {
        super.init(context);
        this.renderTarget.addClass("target");
        if (jQuery("#weather").length === 0) { // build structure
            jQuery("<span/>").attr({id: "name"}).appendTo(this.renderTarget);
            jQuery("<br/>").appendTo(this.renderTarget);
            jQuery("<span/>").attr({id: "weather"}).appendTo(this.renderTarget);
            jQuery("<img/>").attr({id: "icon", src: ""}).appendTo(this.renderTarget);
            jQuery("<br/>").appendTo(this.renderTarget);
            jQuery("<span/>").attr({id: "condition"}).appendTo(this.renderTarget);
        }
    }

    show() {
        const uri = this.baseURI + "?q=" + this.location.trim() + "&APPID=" + this.key;
        const timeReference = Date.now(); // take reference time
        jQuery.ajax({
            url: uri,
            crossDomain: true
        }).then(function (data) {
            this.responseTime = Date.now() - timeReference; // take 2nd reference time, compute delta
            this.log.debug("Current service response time is " + this.responseTime + " ms");
            this.handleWeather(data);
        }.bind(this));
    }

    handleWeather(data) {
        const condition = data.weather[0].main || "";
```

```
47          const icon = this.iconURI + data.weather[0].icon + ".png";
48          const temperature = parseFloat(data.main.temp) - 273.15;
49          this.currentTemperature = temperature.toFixed(1);
50          jQuery("#name").text(data.name);
51          jQuery("#weather").text(this.currentTemperature + " °C");
52          jQuery("#condition").text(condition);
53          jQuery("#icon").attr({src: icon});
54      }
55
56      invokeOperation(name, parameters) {
57          switch (name) {
58              case "update":
59                  this.setProperty("location", parameters.getBody()["location"]);
60                  break;
61              default:
62                  this.log.warn(OpenWeatherMapWeather.name, "Unhandled operation " + name + " invoked.");
63          }
64      }
65
66      setProperty(name, value) {
67          super.setProperty(name, value);
68          switch (name) {
69              case "location":
70                  this.location = value;
71                  this.show();
72                  break;
73              case "ResponseTime":
74                  break;
75              case "CurrentTemperature":
76                  break;
77          }
78      }
79
80      getProperty(name) {
81          const value = super.getProperty(name);
82          if (value != null) return value;
83          switch (name) {
84              case "location":
85                  return this.location;
86              case "ResponseTime":
87                  return this.responseTime;
88              case "CurrentTemperature":
89                  return this.currentTemperature;
90          }
91      }
92  }
```

Codeausschnitt C.2: Vorlage für die Implementierung von Mashup-Komponenten in JavaScript mit Fallbacks für den Lebenszyklus

```
"use strict";
/**
 * Default implementation for a Web mashup component to be subclassed.
 */
class MashupComponent {
    init(context) {
        this.renderTarget = jQuery("#" + context.getAttribute("renderTargetId"));
        this.log = context.getAttribute("Logger");
        this.proxy = context.getAttribute("ServiceAccess");
        this.eventhandler = context.getAttribute("EventHandler");
    }

    setProperty(name, value) {
        switch (name) {
            case "title":
                this.title = value;
                break;
            case "width":
                this.width = parseInt(value);
                break;
            case "height":
                this.height = parseInt(value);
                break;
        }
    }

    getProperty(name) {
        switch (name) {
            case "title":
                return this.title;
            case "width":
                return this.width;
            case "height":
                return this.height;
            default:
                return null;
        }
    }

    show() {
        this.log.warn(MashupComponent.name, "show method not implemented!");
    }

    invokeOperation(name, parameters) {
        this.log.warn(MashupComponent.name, "invokeOperation method not implemented!");
    }
}
```

Anhang D

Werkzeuge

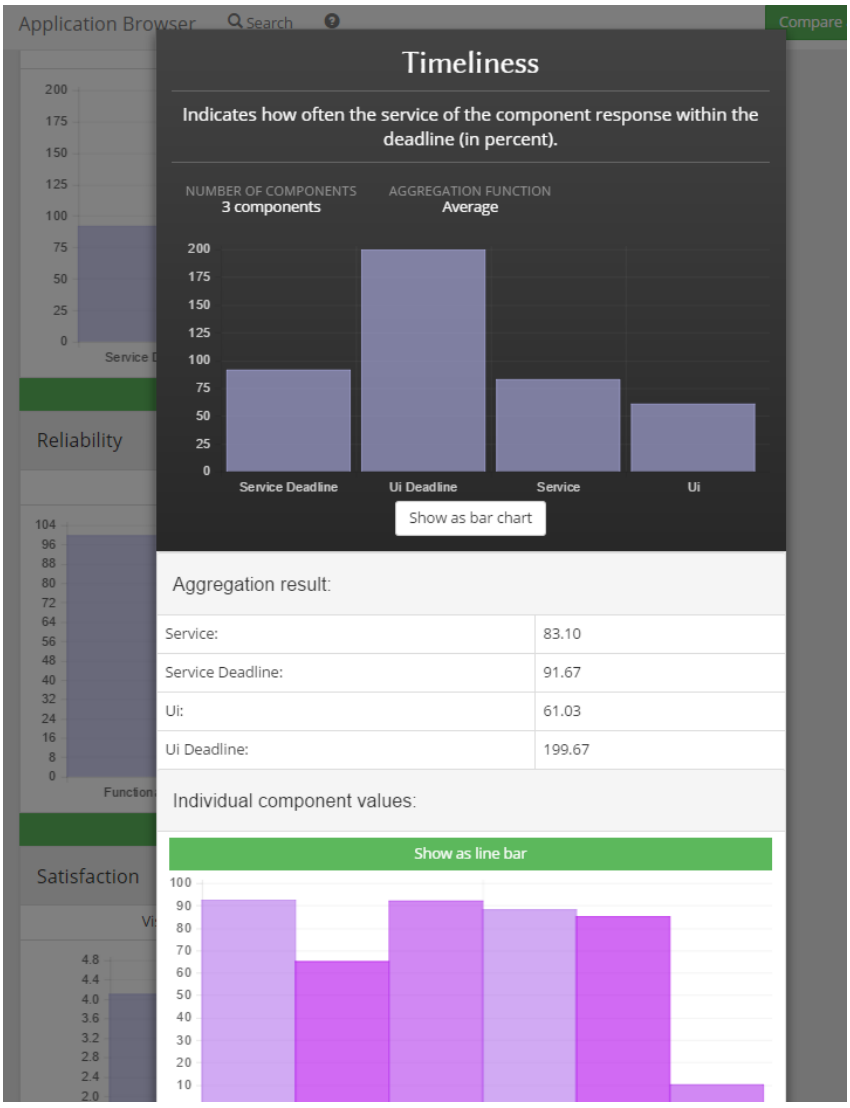

Abbildung D.1: Anwendungsbrowser: Detailansicht zur Aggregation verschiedener Werte der Qualitätseigenschaften

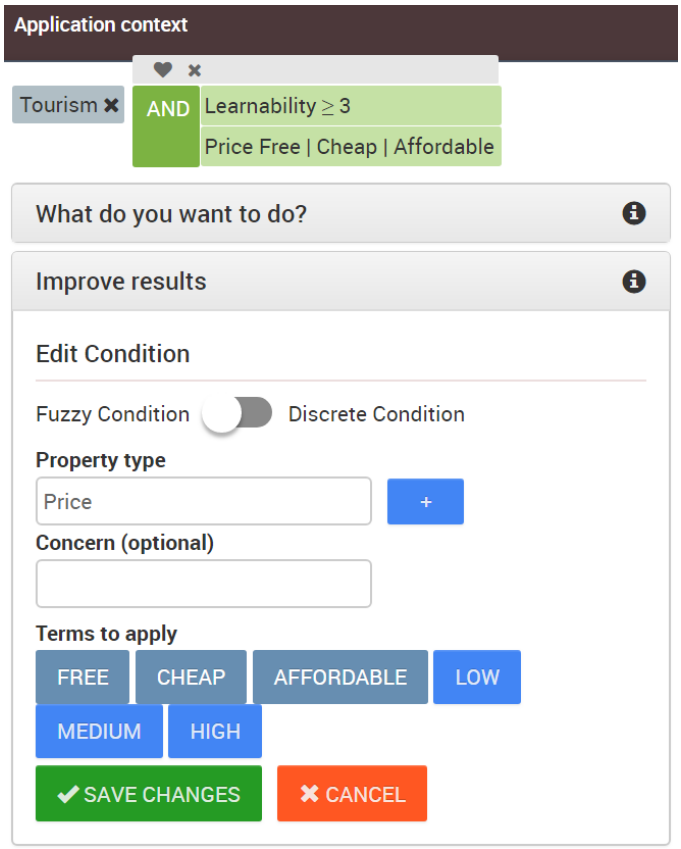

Abbildung D.2: Condition-Editor mit Fuzzy-Bedingung und als Favorit markierter Anforderung

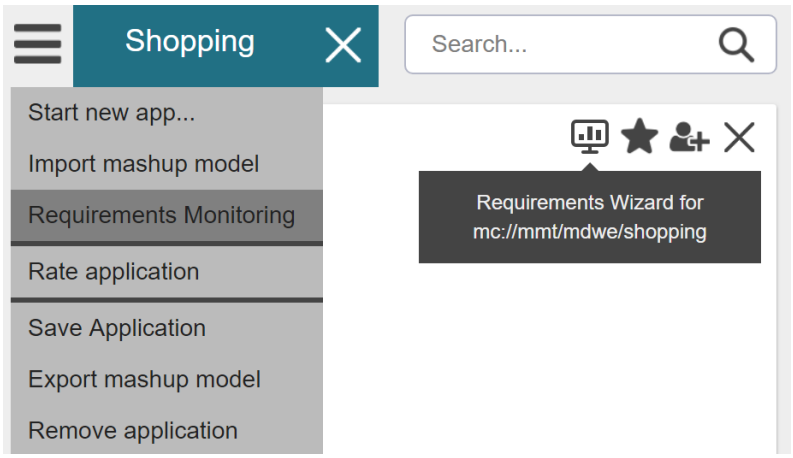

Abbildung D.3: Aktivieren des Anforderungsassistenten im Überwachungsmodus für verschiedene Scopes: *Anwendung* (links über Plattform-Menü) und *Komponente* (rechts über Titelleisten-Icon)

Anhang E

Dienste, Verwaltung und Tests

Codeausschnitt E.1: Ausschnitt aus einem Unit-Test für das automatisierte Setup und Auswerten von Qualitätsanforderungen

```
1  /**
2   * Unit test for user requirements evaluation of composition fragments aka. Web mashup components.
3   * @see Evaluator
4   */
5  public class RequirementsEvaluationTest {
6      private static String componentID = "mc://mmt/mdwe/places";
7      private static String componentURI = "https://example.com/components/trunk/UI_Places/Places.smcdl";
8      private static String component = ClientBuilder.newClient().target(componentURI).request().get().
           readEntity(String.class);
9      private static List<Candidate> candidates = new ArrayList<>();
10
11     /**
12      * Registers candidate component in repository.
13      */
14     @BeforeClass
15     public static void setUpComponents() throws IOException {
16         Core.getCoreObject(ResourceBundle.getBundle("config"), false);
17         McdlManager.registerOrUpdateComponent(component);
18     }
19
20     /**
21      * Tests a requirement with discrete condition and multi-part feature path.
22      */
23     @Test
24     public void discreteDeepFeaturePath() {
25         Individual condition = prepareCondition(QualityRequirement.DiscreteCondition);
26         String featurePath = QualityPropertyReference.Maintainer.getLocalName() + "/" +
               QualityPropertyReference.Name.getLocalName();
27         condition.addProperty(QualityRequirement.onFeature, featurePath);
28         condition.addProperty(QualityRequirement.hasOperator, QualityRequirement.Contains);
29         condition.addLiteral(QualityRequirement.hasCompareValue, "Andreas");
30         assertQualityRating(1.0, condition.getOntModel());
31     }
32
33     /**
34      * Tests a requirement with composite condition having one fuzzy condition and one discrete condition
            as leaves.
35      */
36     @Test
37     public void compositeFuzzyAndDiscrete() {
38         // build conditions
39         Individual cc = prepareCondition(QualityRequirement.CompositeCondition);
40         Individual dc = cc.getOntModel().createIndividual(QualityRequirement.DiscreteCondition);
41         Individual fc = cc.getOntModel().createIndividual(QualityRequirement.FuzzyCondition);
42
43         // build tree, connective (and weights)
```

```
44          cc.addProperty(QualityRequirement.hasCondition, dc);
45          cc.addProperty(QualityRequirement.hasCondition, fc);
46          cc.addProperty(QualityRequirement.hasConnective, QualityRequirement.And);
47
48          // fill discrete condition
49          dc.addProperty(QualityRequirement.onFeature, QualityPropertyReference.Version.getLocalName());
50          dc.addProperty(QualityRequirement.hasOperator, QualityRequirement.LessThan);
51          dc.addLiteral(QualityRequirement.hasCompareValue, 50);
52
53          // fill fuzzy condition
54          prepareReferenceData(cc.getOntModel());
55          fc.addProperty(QualityRequirement.onFeature, QualityPropertyReference.Version.getLocalName());
56          fc.addProperty(QualityRequirement.onTerm, QualityPropertyReference.Medium);
57
58          assertQualityRating(0.5, cc.getOntModel());
59      }
60
61      /**
62       * Tests a requirement with a discrete condition and a concern provided.
63       */
64      @Test
65      public void discreteConcern() {
66          Individual condition = prepareCondition(QualityRequirement.DiscreteCondition);
67          condition.addProperty(QualityRequirement.onFeature, QualityPropertyReference.Credibility.
                getLocalName());
68          condition.addProperty(QualityRequirement.onConcern, QualityProperty.UIData);
69          condition.addProperty(QualityRequirement.hasOperator, QualityRequirement.EqualTo);
70          condition.addLiteral(QualityRequirement.hasCompareValue, Boolean.TRUE);
71          prepareReferenceData(condition.getOntModel());
72          assertQualityRating(1.0, condition.getOntModel());
73      }
74
75      /**
76       * Tests a requirement with a fuzzy condition having multiple terms assigned.
77       */
78      @Test
79      public void fuzzyMultipleTerms() {
80          Individual condition = prepareCondition(QualityRequirement.FuzzyCondition);
81          condition.addProperty(QualityRequirement.onFeature, QualityPropertyReference.Version.getLocalName
                ());
82          condition.addProperty(QualityRequirement.onTerm, QualityPropertyReference.Medium);
83          condition.addProperty(QualityRequirement.onTerm, QualityPropertyReference.High);
84          prepareReferenceData(condition.getOntModel());
85          assertQualityRating(1.0, condition.getOntModel());
86      }
87
88      /**
89       * Tests a requirement provided as TURTLE string. Files are addressed relative to test resources
              according to Maven directory layout.
90       *
91       * @throws IOException thrown on missing requirements input TTL file
92       */
93      @Test
94      public void turtle() throws IOException {
95          try (InputStream ttl = this.getClass().getResourceAsStream("/requirements/requirement-multi.ttl"))
                {
96              for (Candidate candidate : new Evaluator(candidates).evaluate(IOUtils.toString(ttl), null)) {
97                  if (candidate.getFragmentID().equals(componentID)) {
98                      Assert.assertEquals(1.0, candidate.getCriteria(CriteriaType.QUALITY_MATCH), 0.0);
99                  }
100             }
101         }
102     }
```

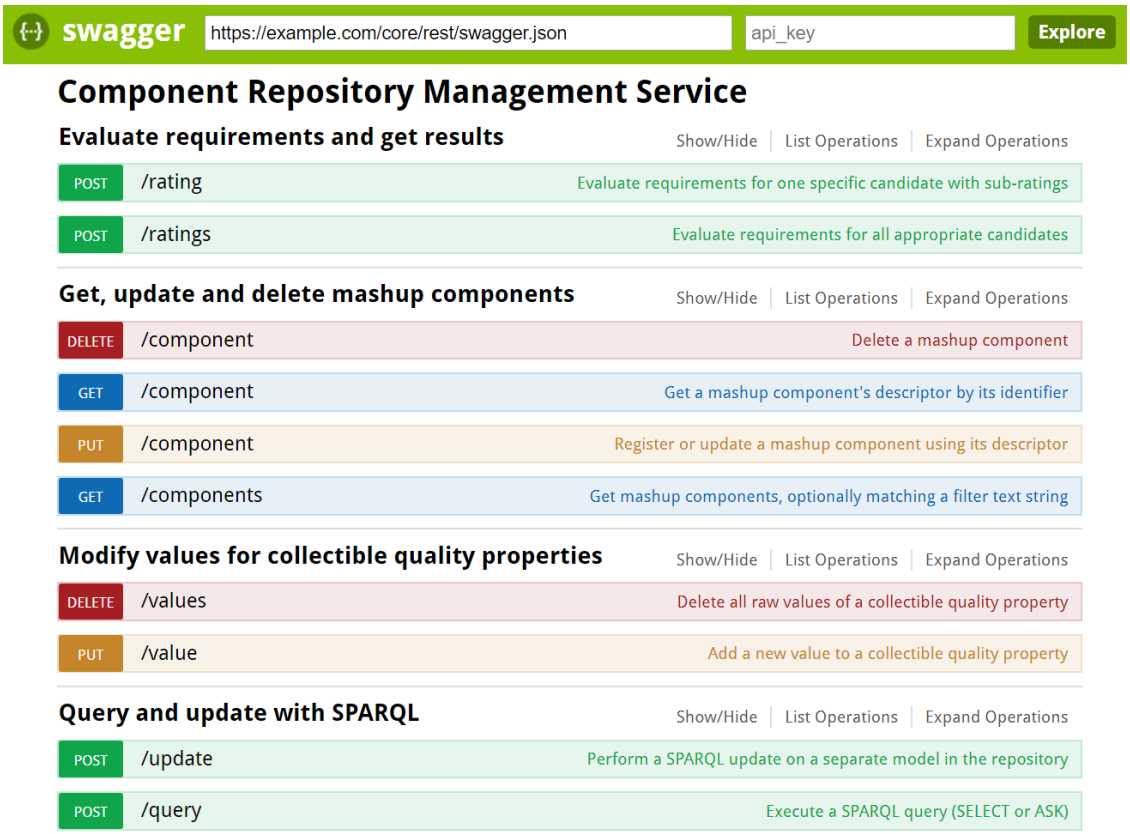

Abbildung E.1: Interaktives Browsen und Erproben der Bestandteile des REST-APIs im Component Repository (CoRe) mit Swagger-UI

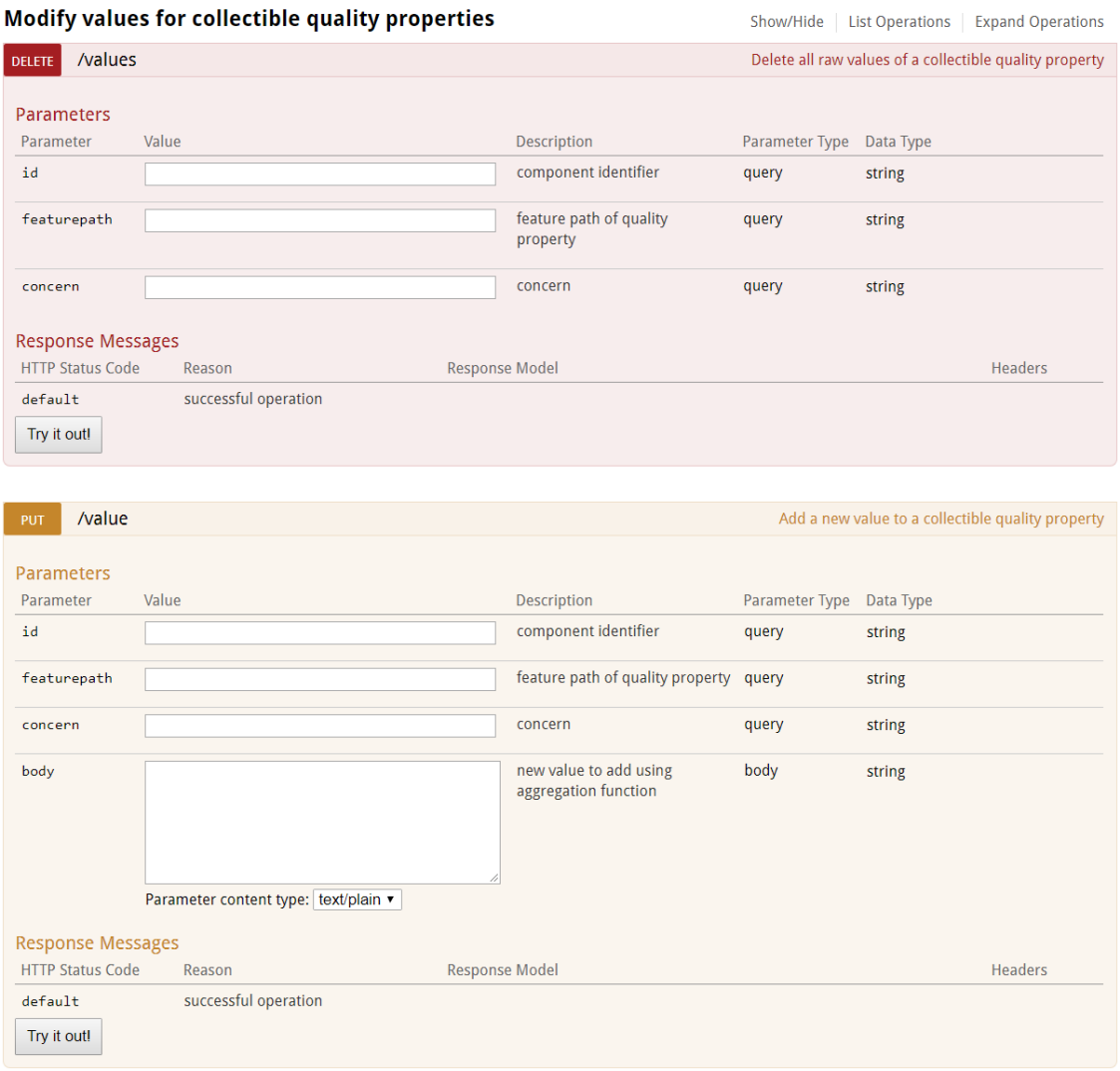

Abbildung E.2: Ausschnitt aus dem Swagger-UI des *REST*-basierten Web-Service zur Manipulation der Werte gesammelter Qualitätseigenschaften

Verzeichnis eigener wissenschaftlicher Veröffentlichungen

[Bli+17] Gregor Blichmann, Andreas Rümpel, Martin Schrader u. a. »Private Data in Collaborative Web Mashups«. In: *Proceedings of the 13th International Conference on Web Information Systems and Technologies (WEBIST 2017)*. Herausgegeben von Tim A. Majchrzak, Karl-Heinz Krempels und Valérie Monfort. INSTICC. ScitePress, April 2017, Seiten 174–183. ISBN: 978-989-758-246-2. DOI: 10.5220/0006374201740183. URL: http://scitepress.org/DigitalLibrary/PublicationsDetail.aspx?ID=F66vuZ/ICZY=.

[Hab+08] Dirk Habich, Sebastian Richly, Andreas Rümpel u. a. »Open Service Process Platform 2.0«. In: *Proceedings of the 2008 IEEE Congress on Services – Part I*. IEEE Computer Society, Juli 2008, Seiten 152–159. ISBN: 978-0-7695-3286-8. DOI: 10.1109/SERVICES-1.2008.27. URL: https://ieeexplore.ieee.org/document/4578318.

[Lor+11] Alexander Lorz, Andreas Rümpel, Carsten Radeck u. a. »Introducing the EDYRA Vision: Engineering of Do-It-Yourself Rich Internet Applications«. In: *Proceedings of the International Conference on Internet Technologies & Society 2011 (ITS 2011)*. Herausgegeben von Piet Kommers, Ji-ping Zhang, Tomayess Issa u. a. IADIS Press, Dezember 2011, Seiten 330–332. ISBN: 978-972-8939-55-7. URL: http://www.iadisportal.org/digital-library/introducing-the-edyra-vision-engineering-of-do-it-yourself-rich-internet-applications.

[Pie+09] Stefan Pietschmann, Martin Voigt, Andreas Rümpel u. a. »CRUISe: Composition of Rich User Interface Services«. In: *Web Engineering*. Herausgegeben von Martin Gaedke, Michael Grossniklaus und Oscar Díaz. Band 5648. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Juni 2009, Seiten 473–476. ISBN: 978-3-642-02817-5. DOI: 10.1007/978-3-642-02818-2_41. URL: https://link.springer.com/chapter/10.1007/978-3-642-02818-2_41.

[RBM10] Andreas Rümpel, Ken Baumgärtel und Klaus Meißner. »Platform Adaptation of Mashup UI Components«. In: *Proceedings of The Second International Conference on Adaptive and Self-adaptive Systems and Applications*. Herausgegeben von Ali Beklen, Jorge Ejarque, Wolfgang Gentzsch u. a. Xpert Publishing Services, November 2010, Seiten 164–169. ISBN: 978-1-61208-109-0. URL: http://www.thinkmind.org/index.php?view=article&articleid=adaptive_2010_7_20_40111.

[RM12] Andreas Rümpel und Klaus Meißner. »Requirements-Driven Quality Modeling and Evaluation in Web Mashups«. In: *2012 Eighth International Conference on the Quality of Information and Communications Technology*. Herausgegeben von João Carlos Pascoal Faria, Alberto Manuel Rodrigues da Silva und Ricardo J. Machado. Los Alamitos, CA, USA: IEEE Computer Society, September 2012, Seiten 319–322. ISBN: 978-1-4673-2345-1. DOI: 10.1109/QUATIC.2012.51. URL: https://ieeexplore.ieee.org/document/6511835.

[Rüm+11] Andreas Rümpel, Carsten Radeck, Gregor Blichmann u. a. »Towards Do-It-Yourself Development of Composite Web Applications«. In: *Proceedings of the International Conference on Internet Technologies & Society 2011 (ITS 2011)*. Herausgegeben von Piet Kommers, Jiping Zhang, Tomayess Issa u. a. IADIS Press, Dezember 2011, Seiten 231–235. ISBN: 978-972-8939-55-7. URL: http://www.iadisportal.org/digital-library/towards-do-it-yourself-development-of-composite-web-applications.

[Rüm+13] Andreas Rümpel, Vincent Tietz, Anika Wagner u. a. »Modeling and Utilizing Quality Properties in the Development of Composite Web Mashups«. In: *Current Trends in Web Engineering*. Herausgegeben von Quan Z. Sheng und Jesper Kjeldskov. Band 8295. Lecture Notes in Computer Science. Springer International Publishing, Juli 2013, Seiten 54–65. ISBN: 978-3-319-04244-2. DOI: 10.1007/978-3-319-04244-2_7. URL: https://link.springer.com/chapter/10.1007/978-3-319-04244-2_7.

[Rüm+14] Andreas Rümpel, Carsten Radeck, Juri Tichomirow u. a. »Fuzzy Mashup Quality Requirements Specification for Web Users«. In: *2014 Ninth International Conference on the Quality of Information and Communications Technology*. Herausgegeben von Alberto Manuel Rodrigues da Silva, António Rito Silva, Miguel Abrunhosa de Brito u. a. IEEE Computer Society, September 2014, Seiten 262–267. ISBN: 978-1-4799-6133-7. DOI: 10.1109/QUATIC.2014.42. URL: https://ieeexplore.ieee.org/document/6984131.

[Tie+12] Vincent Tietz, Andreas Rümpel, Christian Liebing u. a. »Towards Requirements Engineering for Mashups: State of the Art and Research Challenges«. In: *Proceedings of the 7th International Conference on Internet and Web Applications and Services (ICIW 2012)*. Herausgegeben von Friedrich Laux und Pascal Lorenz. Xpert Publishing Services, Mai 2012, Seiten 123–130. ISBN: 978-1-61208-200-4. URL: http://www.thinkmind.org/index.php?view=article&articleid=iciw_2012_4_50_20164.

[Tie+13a] Vincent Tietz, Oliver Mroß, Andreas Rümpel u. a. »A Requirements Model for Composite and Distributed Web Mashups«. In: *Proceedings of the The Eighth International Conference on Internet and Web Applications and Services (ICIW 2013)*. Herausgegeben von Ioannis Moscholios und Marek Rychly. Xpert Publishing Services, Juni 2013, Seiten 75–82. ISBN: 978-1-61208-280-6. URL: http://www.thinkmind.org/index.php?view=article&articleid=iciw_2013_4_10_20161.

[Tie+13b] Vincent Tietz, Andreas Rümpel, Martin Voigt u. a. »Tool Support for Semantic Task Modeling«. In: *Proceedings of the 3rd International Conference on Web Intelligence, Mining and Semantics (WIMS '13)*. Herausgegeben von David Camacho, Maria D. Rodriguez Moreno und Rajendra Akerkar. ACM, Juni 2013, Seiten 1–12. ISBN: 978-1-4503-1850-1. DOI: 10.1145/2479787.2479827. URL: https://dl.acm.org/citation.cfm?id=2479827.

Literaturverzeichnis

[Aag01] Jan Øyvind Aagedal. »Quality of Service Support in Development of Distributed Systems«. PhD Thesis. Universitetet i Oslo, 2001 (siehe Seite 49).

[AP12] Saeed Aghaee und Cesare Pautasso. »EnglishMash: Usability Design for a Natural Mashup Composition Environment«. In: *Current Trends in Web Engineering*. Band 7703. Lecture Notes in Computer Science. Springer, 2012, Seiten 109–120 (siehe Seiten 41, 43).

[Apa15] The Apache Software Foundation. *Architectural overview of Cordova platform*. 2015. URL: https://cordova.apache.org/docs/en/latest/guide/overview/index.html (siehe Seite 14).

[Bal98] Helmut Balzert. *Lehrbuch der Softwaretechnik: Softwaremanagement*. 2. Auflage. Spektrum Akademischer Verlag, Februar 1998. ISBN: 978-3827411617. URL: https://www.amazon.de/dp/3827411610 (siehe Seite 30).

[BMS10] Elisa Bertino, Andrea Maurino und Monica Scannapieco. »Guest Editors' Introduction: Data Quality in the Internet Era«. In: *IEEE Internet Computing* 14.4 (Juli 2010), Seiten 11–13. DOI: 10.1109/MIC.2010.93. URL: https://ieeexplore.ieee.org/document/5496804/ (siehe Seite 43).

[BRM13] Gregor Blichmann, Carsten Radeck und Klaus Meißner. »Enabling End Users to Build Situational Collaborative Mashups at Runtime«. In: *Proceedings of The Eighth International Conference on Internet and Web Applications and Services*. Herausgegeben von Ioannis Mosciolios und Marek Rychly. Xpert Publishing Services, Juni 2013, Seiten 120–123. ISBN: 978-1-61208-280-6. URL: http://www.thinkmind.org/index.php?view=article&articleid=iciw_2013_5_40_20132 (siehe Seiten 103, 122).

[Cap+10] Cinzia Cappiello, Florian Daniel, Maristella Matera u. a. »Information Quality in Mashups«. In: *IEEE Internet Computing* 14.4 (Juli 2010), Seiten 14–22. ISSN: 1089-7801. DOI: 10.1109/MIC.2010.60. URL: https://ieeexplore.ieee.org/document/5453330/ (siehe Seiten 39, 40, 43, 71).

[Cap+11a] Cinzia Cappiello, Florian Daniel, Agnes Koschmider u. a. »A Quality Model for Mashups«. In: *Web Engineering*. Herausgegeben von Sören Auer, Oscar Díaz und George Papadopoulos. Band 6757. Lecture Notes in Computer Science. Springer, Juni 2011, Seiten 137–151. DOI: 10.1007/978-3-642-22233-7_10. URL: https://link.springer.com/chapter/10.1007%2F978-3-642-22233-7_10 (siehe Seiten 37, 39, 76, 145).

[Cap+11b] Cinzia Cappiello, Florian Daniel, Maristella Matera u. a. »Enabling End User Development through Mashups: Requirements, Abstractions and Innovation Toolkits«. In: *End-User Development*. Herausgegeben von Maria Costabile, Yvonne Dittrich, Gerhard Fischer u. a. Band 6654. Lecture Notes in Computer Science. Springer, Juni 2011, Seiten 9–24. DOI: 10.1007/978-3-642-21530-8_3. URL: hhttps://link.springer.com/chapter/10.1007%2F978-3-642-21530-8_3 (siehe Seite 41).

[Cap+11c] Cinzia Cappiello, Maristella Matera, Matteo Picozzi u. a. »DashMash: A Mashup Environment for End User Development«. In: *Web Engineering*. Herausgegeben von Sören Auer, Oscar Díaz und George Papadopoulos. Band 6757. Lecture Notes in Computer Science. Springer, Juni 2011, Seiten 152–166. DOI: 10.1007/978-3-642-22233-7_11. URL: https://link.springer.com/chapter/10.1007%2F978-3-642-22233-7_11 (siehe Seiten 1, 41, 43).

[Cap+12] Cinzia Cappiello, Maristella Matera, Matteo Picozzi u. a. »Quality-Aware Mashup Composition: Issues, Techniques and Tools«. In: *Proceedings of the 8th International Conference on the Quality of Information and Communications Technology (QUATIC 2012)*. Herausgegeben von João Pascoal Faria, Alberto Silva und Ricardo Machado. IEEE Computer Society, September 2012, Seiten 10–19. ISBN: 978-0-7695-4777-0. DOI: 10.1109/QUATIC.2012.50. URL: https://ieeexplore.ieee.org/document/6511777/ (siehe Seiten 41, 42).

[CDM09] Cinzia Cappiello, Florian Daniel und Maristella Matera. »A Quality Model for Mashup Components«. In: *Web Engineering*. Herausgegeben von Martin Gaedke, Michael Grossniklaus und Oscar Díaz. Band 5648. Lecture Notes in Computer Science. Springer, Juni 2009, Seiten 236–250. DOI: 10.1007/978-3-642-02818-2_19. URL: https://link.springer.com/chapter/10.1007%2F978-3-642-02818-2_19 (siehe Seiten 37, 38, 41, 76, 78, 80, 145).

[CFT07] G. Castellano, A. M. Fanelli und M. A. Torsello. »REXWERE: A tool for fuzzy Rule EXtraction in WEb REcommendation«. In: *Fuzzy Information Processing Society, 2007. NAFIPS '07. Annual Meeting of the North American*. 2007, Seiten 129–134. DOI: 10.1109/NAFIPS.2007.383824. URL: https://ieeexplore.ieee.org/document/4271047/ (siehe Seite 33).

[Chu+13] Olexiy Chudnovskyy, Stefan Pietschmann, Matthias Niederhausen u. a. »Awareness and Control for Inter-Widget Communication: Challenges and Solutions«. In: *Web Engineering*. Herausgegeben von Florian Daniel, Peter Dolog und Qing Li. Band 7977. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Juli 2013, Seiten 114–122. ISBN: 978-3-642-39200-9. DOI: 10.1007/978-3-642-39200-9_11. URL: https://link.springer.com/chapter/10.1007%2F978-3-642-39200-9_11 (siehe Seiten 28, 109, 151).

[CL07] Yukun Cao und Yunfeng Li. »An intelligent fuzzy-based recommendation system for consumer electronic products«. In: *Expert Systems with Applications* 33.1 (2007), Seiten 230–240. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2006.04.012. URL: https://www.sciencedirect.com/science/article/pii/S0957417406001369 (siehe Seite 32).

[CP09] Marco Comuzzi und Barbara Pernici. »A framework for QoS-based Web service contracting«. In: *ACM Transactions on the Web* 3.3 (Juni 2009), Seiten 1–52. ISSN: 15591131. DOI: 10.1145/1541822.1541825. URL: https://dl.acm.org/citation.cfm?id=1541825 (siehe Seite 47).

[CX10] Jiawei Cao und Chunxiao Xing. »Data Source Recommendation for Building Mashup Applications«. In: *2010 Seventh Web Information Systems and Applications Conference*. 2009. IEEE Computer Society, August 2010, Seiten 220–224. ISBN: 978-1-4244-8440-9. DOI: 10.1109/WISA.2010.39. URL: https://ieeexplore.ieee.org/document/5581363/ (siehe Seite 41).

[DAI10] Oscar Díaz, Cristóbal Arellano und Jon Iturrioz. »Interfaces for Scripting: Making Greasemonkey Scripts Resilient to Website Upgrades«. In: *Web Engineering*. Herausgegeben von Boualem Benatallah, Fabio Casati, Gerti Kappel u. a. Band 6189. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Juli 2010, Seiten 233–247. ISBN: 978-3-642-13911-6. DOI: 10.1007/978-3-642-13911-6_16. URL: https://link.springer.com/chapter/10.1007/978-3-642-13911-6_16 (siehe Seite 17).

[Elm+08] Hazem Elmeleegy, Anca Ivan, Rama Akkiraju u. a. »Mashup Advisor: A Recommendation Tool for Mashup Development«. In: *2008 IEEE International Conference on Web Services*. IEEE Computer Society, September 2008, Seiten 337–344. DOI: 10.1109/ICWS.2008.128. URL: https://ieeexplore.ieee.org/document/4670193/ (siehe Seite 42).

[Fau03] Laura Faulkner. »Beyond the five-user assumption: Benefits of increased sample sizes in usability testing«. In: *Behavior Research Methods, Instruments, & Computers* 35.3 (2003), Seiten 379–383. ISSN: 1532-5970. DOI: 10.3758/BF03195514. URL: https://link.springer.com/article/10.3758/BF03195514 (siehe Seite 136).

[GC87] Robert B. Grady und Deborah L. Caswell. *Software Metrics: Establishing a Company-Wide Program*. Prentice Hall, 6. Juni 1987. ISBN: 0138218447 (siehe Seite 76).

[Göt+12] Sebastian Götz, Claas Wilke, Sebastian Cech u. a. »Architecture and Mechanisms for Energy Auto Tuning«. In: *Sustainable ICTs and Management Systems for Green Computing*. 2012, Seiten 45–73. ISBN: 9781466618398. DOI: 10.4018/978-1-4666-1839-8.ch003. URL: http://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa-116748 (siehe Seite 49).

[Gov09] Government 2.0 Taskforce. *Engage – Getting on with Government 2.0 | Report of the Government 2.0 Taskforce*. Dezember 2009. URL: http://www.finance.gov.au/sites/default/files/Government20TaskforceReport.pdf?v=1 (siehe Seite 1).

[GS10] Lars Grammel und Margaret-Anne Storey. »A Survey of Mashup Development Environments«. In: *The Smart Internet*. Band 6400. Lecture Notes in Computer Science. Springer, 2010, Seiten 137–151 (siehe Seite 21).

[Hit+08] Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph u. a. *Semantic Web – Grundlagen*. Springer-Verlag Berlin Heidelberg, 2008. ISBN: 978-3-540-33993-9. DOI: 10.1007/978-3-540-33994-6 (siehe Seite 15).

[HM95] Abdollah Homaifar und Ed McCormick. »Simultaneous Design of Membership Functions and Rule Sets for Fuzzy Controllers Using Genetic Algorithms«. In: *IEEE Transactions on Fuzzy Systems* 3.2 (Mai 1995), Seiten 129–139. URL: https://ieeexplore.ieee.org/document/388168/ (siehe Seite 33).

[Hod+14] Ralph Hodgson, Paul J. Keller, Jack Hodges u. a. *QUDT – Quantities, Units, Dimensions and Data Types Ontologies*. März 2014. URL: http://qudt.org (siehe Seiten 15, 55).

[ISO25010] International Organization for Standardization. *ISO/IEC IS 25010:2011: Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. ISO/IEC, März 2011 (siehe Seiten 28, 76).

[ISO25030] International Organization for Standardization. *ISO/IEC IS 25030:2007: Software engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Quality requirements*. ISO/IEC, 2007 (siehe Seite 29).

[ISO80000] International Organization for Standardization. *ISO IS 80000:2008: Quantities and units – Part 11: Characteristic numbers*. ISO, 2008 (siehe Seite 72).

[ISO9126] International Organization for Standardization. *ISO/IEC 9126-1:2001: Software engineering – Product quality – Part 1: Quality model*. ISO/IEC, 2001 (siehe Seiten 28, 38, 76).

[KKA12] Keivan Kianmehr, Negar Koochakzadeh und Reda Alhajj. »AskFuzzy: Attractive Visual Fuzzy Query Builder«. In: *2012 IEEE 28th International Conference on Data Engineering (ICDE '12)*. IEEE Computer Society, 2012, Seiten 1241–1244. ISBN: 978-0-7695-4747-3. DOI: 10.1109/ICDE.2012.116. URL: https://dl.acm.org/citation.cfm?id=2310344 (siehe Seiten 32, 33).

[Lee+03] KangChan Lee, JongHong Jeon, WonSeok Lee u. a. *QoS for Web Services: Requirements and Possible Approaches*. Herausgegeben von KangChan Lee, JongHing Jeon, WonSeok Lee u. a. 25. November 2003. URL: http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/ (siehe Seite 76).

[Loc10] Klaus Lochmann. »Engineering Quality Requirements Using Quality Models«. In: *15th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE, März 2010, Seiten 245–246. ISBN: 978-1-4244-6639-9. DOI: 10.1109/ICECCS.2010.24. URL: https://ieeexplore.ieee.org/document/5628608/ (siehe Seite 31).

[Loh+16] Steffen Lohmann, Stefan Negru, Florian Haag u. a. »Visualizing Ontologies with VOWL«. In: *Semantic Web* 7.4 (2016), Seiten 399–419. DOI: 10.3233/SW-150200. URL: https://content.iospress.com/articles/semantic-web/sw200 (siehe Seite 58).

[LOZ10] Philip Lew, Luis Olsina und Li Zhang. »Integrating Quality, Quality in Use, Actual Usability and User Experience«. In: *6th Central and Eastern European Software Engineering Conference (CEE-SECR)*. 978. IEEE Computer Society, 2010, Seiten 117–123 (siehe Seiten 37, 44).

[Mat+13] Maristella Matera, Matteo Picozzi, Michele Pini u. a. »PEUDOM: A Mashup Platform for the End User Development of Common Information Spaces«. In: *Web Engineering (ICWE 2013)*. Band 7977. Lecture Notes in Computer Science. Springer, Juni 2013, Seiten 494–497. URL: https://link.springer.com/chapter/10.1007/978-3-642-39200-9_43 (siehe Seiten 41, 43).

[MM14] Oliver Mroß und Klaus Meißner. »Towards User-Centered Distributed Mashups«. In: *Proceedings of the 4th Workshop on Distributed User Interfaces and Multimodal Interaction (DUI 2014)*. ACM, Juli 2014, Seiten 11–14. ISBN: 978-1-60558-724-0. DOI: 10.1145/2677356.2677658. URL: https://dl.acm.org/citation.cfm?id=2677658 (siehe Seiten 96, 100, 103, 113, 120, 128, 150, 171).

[MPS02] Giulio Mori, Fabio Paternò und Carmen Santoro. »CTTE: Support for Developing and Analyzing Task Models for Interactive System Design«. In: *IEEE Transactions on Software Engineering* 28.8 (August 2002), Seiten 797–813. ISSN: 0098-5589. DOI: 10.1109/TSE.2002.1027801. URL: https://ieeexplore.ieee.org/document/1027801/ (siehe Seite 32).

[MPT08] Fernando Molina, Jesús Pardillo und Ambrosio Toval. »Modelling Web-Based Systems Requirements Using WRM«. In: *Web Information Systems Engineering – Workshops*. Herausgegeben von Sven Hartmann, Xiaofang Zhou und Markus Kirchberg. Band 5176. Lecture Notes in Computer Science. Springer, September 2008, Seiten 122–131. DOI: 10.1007/978-3-540-85200-1_14. URL: https://link.springer.com/chapter/10.1007/978-3-540-85200-1_14 (siehe Seite 45).

[MRW77] Jim A. McCall, Paul K. Richards und Gene F. Walters. *Factors in Software Quality – Concept and Definitions of Software Quality*. Technischer Bericht. General Electric Company, Juli 1977. URL: http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA049014 (siehe Seite 76).

[MS14] Microsoft Corporation. *Developing Windows Store apps (HTML)*. 2014. URL: http://msdn.microsoft.com/en-us/library/windows/apps/br229565 (siehe Seite 14).

[Obe+13] Daniel Oberle, Alistair Barros, Uwe Kylau u. a. »A unified description language for human to automated services«. In: *Information Systems* 38.1 (März 2013), Seiten 155–181. ISSN: 03064379. DOI: 10.1016/j.is.2012.06.004. URL: https://www.sciencedirect.com/science/article/pii/S0306437912000968 (siehe Seite 47).

[OEH05] Justin O'Sullivan, David Edmond und Arthur H. M. ter Hofstede. *Formal description of non-functional service properties*. Technischer Bericht. Brisbane: Queensland University of Technology, Februar 2005. URL: http://www.wsmo.org/papers/OSullivanTR2005.pdf (siehe Seiten 37, 47).

[Ols+12] Luis Olsina, Philip Lew, Alexander Dieser u. a. »Updating Quality Models for Evaluating New Generation Web Applications«. In: *Journal of Web Engineering* 11.3 (2012), Seiten 209–246. URL: https://dl.acm.org/citation.cfm?id=2481554 (siehe Seiten 37, 44, 45).

[Ore10] T. Orehovacki. »Proposal for a set of quality attributes relevant for Web 2.0 application success«. In: *2010 32nd International Conference on Information Technology Interfaces (ITI)*. 2010, Seiten 319–326 (siehe Seite 76).

[OSM08] Luis Olsina, Roberto Sassano und Luisa Mich. »Specifying Quality Requirements for the Web 2.0 Applications«. In: *Web Engineering*. Lecture Notes in Computer Science. Springer, 2008, Seiten 56–62 (siehe Seite 76).

[Pap+06] I. V. Papaioannou, D. T. Tsesmetzis, I. G. Roussaki u. a. »A QoS ontology language for Web-services«. In: *20th International Conference on Advanced Information Networking and Applications – Volume 1 (AINA'06)*. IEEE, 2006, Seiten 101–106. ISBN: 0-7695-2466-4. DOI: 10.1109/AINA.2006.51. URL: https://ieeexplore.ieee.org/document/1620177/ (siehe Seite 76).

[Pic+10] Matteo Picozzi, Marta Rodolfi, Cinzia Cappiello u. a. »Quality-based Recommendations for Mashup Composition«. In: *Web Engineering*. Herausgegeben von Florian Daniel und Federico Facca. Band 6385. Lecture Notes in Computer Science. Springer, Juli 2010, Seiten 360–371. DOI: 10.1007/978-3-642-16985-4_32. URL: https://link.springer.com/chapter/10.1007%2F978-3-642-16985-4_32 (siehe Seiten 39, 43).

[Pie+09] Stefan Pietschmann, Martin Voigt, Andreas Rümpel u. a. »CRUISe: Composition of Rich User Interface Services«. In: *Web Engineering*. Herausgegeben von Martin Gaedke, Michael Grossniklaus und Oscar Díaz. Band 5648. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Juni 2009, Seiten 473–476. ISBN: 978-3-642-02817-5. DOI: 10.1007/978-3-642-02818-2_41. URL: https://link.springer.com/chapter/10.1007/978-3-642-02818-2_41 (siehe Seite 15).

[Pie12] Stefan Pietschmann. »Modellgetriebene Entwicklung adaptiver, komponentenbasierter Mashup-Anwendungen«. Dissertation. Technische Universität Dresden, Dezember 2012. URL: http://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa-101559 (siehe Seiten 13, 15, 22, 24, 110, 123, 136).

[Pol09] Jan Polowinski. »Widgets for Faceted Browsing«. In: *Human Interface and the Management of Information. Designing Information Environments*. LNCS 5617 (2009), Seiten 601–610. DOI: 10.1007/978-3-642-02556-3_68. URL: https://link.springer.com/chapter/10.1007%2F978-3-642-02556-3_68 (siehe Seite 32).

[Pou09] Michael Poulin. *Ladder to SOE: How to Create Resourceful and Efficient Solutions for Market Changes Within Business and Technology*. Troubador Pub Ltd, August 2009. ISBN: 978-1848761629. URL: https://www.amazon.de/dp/1848761627 (siehe Seite 1).

[Pro11] ProgrammableWeb.com. *Keeping you up to date with APIs, mashups and the Web as platform*. 2011. URL: https://www.programmableweb.com (siehe Seiten 1, 41).

[Rad+12] Carsten Radeck, Alexander Lorz, Gregor Blichmann u. a. »Hybrid Recommendation of Composition Knowledge for End User Development of Mashups«. In: *The Seventh International Conference on Internet and Web Applications and Services (ICIW 2012)*. Xpert Publishing Services, 2012, Seiten 30–33. ISBN: 9781612082004. URL: http://www.thinkmind.org/index.php?view=article%5C&articleid=iciw_2012_2_10_20180 (siehe Seiten 9, 42, 92, 105, 122, 127, 142, 150).

[Rad+14] Carsten Radeck, Gregor Blichmann, Oliver Mroß u. a. »Semantic Mediation Techniques for Composite Web Applications«. In: *Web Engineering*. Herausgegeben von Sven Casteleyn, Marco Winckler und Gustavo Rossi. Band 8541. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2014, Seiten 450–459 (siehe Seiten 9, 43, 78).

[Ran03] Shuping Ran. »A model for web services discovery with QoS«. In: *ACM SIGecom Exchanges* 4.1 (März 2003), Seiten 1–10. ISSN: 15519031. DOI: 10.1145/844357.844360 (siehe Seite 76).

[RBM16] Carsten Radeck, Gregor Blichmann und Klaus Meißner. »Estimating the Functionality of Mashup Applications for Assisted, Capability-centered End User Development«. In: *Proceedings of the 12th International Conference on Web Information Systems and Technologies (WEBIST 2016)*. April 2016, Seiten 109–120. ISBN: 978-989-758-186-1. URL: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0005802601090120 (siehe Seiten 85, 105, 120).

[RM03] Rita A. Ribeiro und Ana M. Moreira. »Fuzzy query interface for a business database«. In: *International Journal of Human-Computer Studies* 58.4 (2003), Seiten 363–391. ISSN: 1071-5819. DOI: 10.1016/S1071-5819(03)00010-7. URL: https://www.sciencedirect.com/science/article/pii/S1071581903000107 (siehe Seite 32).

[RM12] Andreas Rümpel und Klaus Meißner. »Requirements-Driven Quality Modeling and Evaluation in Web Mashups«. In: *2012 Eighth International Conference on the Quality of Information and Communications Technology*. Herausgegeben von João Carlos Pascoal Faria, Alberto Manuel Rodrigues da Silva und Ricardo J. Machado. Los Alamitos, CA, USA: IEEE Computer Society, September 2012, Seiten 319–322. ISBN: 978-1-4673-2345-1. DOI: 10.1109/QUATIC.2012.51. URL: https://ieeexplore.ieee.org/document/6511835 (siehe Seite 110).

[Rüm+11] Andreas Rümpel, Carsten Radeck, Gregor Blichmann u. a. »Towards Do-It-Yourself Development of Composite Web Applications«. In: *Proceedings of the International Conference on Internet Technologies & Society 2011 (ITS 2011)*. Herausgegeben von Piet Kommers, Jiping Zhang, Tomayess Issa u. a. IADIS Press, Dezember 2011, Seiten 231–235. ISBN: 978-972-8939-55-7. URL: http://www.iadisportal.org/digital-library/towards-do-it-yourself-development-of-composite-web-applications (siehe Seiten 16, 17, 43, 121).

[Rüm+13] Andreas Rümpel, Vincent Tietz, Anika Wagner u. a. »Modeling and Utilizing Quality Properties in the Development of Composite Web Mashups«. In: *Current Trends in Web Engineering*. Herausgegeben von Quan Z. Sheng und Jesper Kjeldskov. Band 8295. Lecture Notes in Computer Science. Springer International Publishing, Juli 2013, Seiten 54–65. ISBN: 978-3-319-04244-2. DOI: 10.1007/978-3-319-04244-2_7. URL: https://link.springer.com/chapter/10.1007/978-3-319-04244-2_7 (siehe Seite 75).

[Rüm+14] Andreas Rümpel, Carsten Radeck, Juri Tichomirow u. a. »Fuzzy Mashup Quality Requirements Specification for Web Users«. In: *2014 Ninth International Conference on the Quality of Information and Communications Technology*. Herausgegeben von Alberto Manuel Rodrigues da Silva, António Rito Silva, Miguel Abrunhosa de Brito u. a. IEEE Computer

Society, September 2014, Seiten 262–267. ISBN: 978-1-4799-6133-7. DOI: 10.1109/QUATIC.2014.42. URL: https://ieeexplore.ieee.org/document/6984131 (siehe Seite 83).

[Rup11] Chris Rupp. *QAMP – auf dem Weg zum Software Quality Engineering, Teil 2*. Februar 2011. URL: https://www.heise.de/-1189917 (siehe Seite 23).

[RZ03] Simone Röttger und Steffen Zschaler. »CQML⁺: Enhancements to CQML«. In: *Proceedings of the 1st International Workshop on Quality of Service in Component-Based Software Engineering*. Herausgegeben von Jean-Michel Bruel. Cépaduès-Éditions, Juni 2003, Seiten 43–56. ISBN: 978-2854286175. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.58.6672 (siehe Seite 49).

[RZ07] Simone Röttger und Steffen Zschaler. »Tool Support for Refinement of Non-functional Specifications«. In: *Software and Systems Modelling Journal* 6.2 (2007), Seiten 185–204 (siehe Seiten 37, 49).

[RZ74] Floyd L. Ruch und Philip G. Zimbardo. *Lehrbuch der Psychologie. Eine Einführung für Studenten der Psychologie, Medizin und Pädagogik*. Springer-Verlag Berlin Heidelberg, 1974. ISBN: 978-3-662-08328-4. DOI: 10.1007/978-3-662-08328-4. URL: https://link.springer.com/book/10.1007/978-3-662-08328-4 (siehe Seite 105).

[Sch11] Ben Schwan. »Mashups für die Verwaltung«. In: *Technology Review* (März 2011). URL: https://heise.de/-1200468 (siehe Seite 1).

[SPARQL] World Wide Web Consortium. *SPARQL 1.1 Query Language*. Herausgegeben von Steven Harris und Andy Seaborne. W3C Recommendation. W3C, März 2013. URL: https://www.w3.org/TR/sparql11-query/ (siehe Seite 85).

[Szy02] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Second. Addison-Wesley, November 2002. ISBN: 978-0201745726. URL: https://www.amazon.de/dp/0201745720 (siehe Seite 13).

[Ter+12] Luis Terán, Andreas Lander, Jan Fivaz u. a. »Using a Fuzzy-Based Cluster Algorithm for Recommending Candidates in E-Elections«. In: *Digital Democracy: Concepts, Methodologies, Tools, and Applications* (2012), Seiten 684–705. URL: https://diuf.unifr.ch/main/is/sites/diuf.unifr.ch.main.is/files/documents/publications/Using_a_Fuzzy_Based_Cluster_Algorithm_for_Recommending_Candidates_in_eElections.pdf (siehe Seite 32).

[Tho14] Yvonne Thoss. »Systemunterstützung zur Bewertung der Qualität persönlicher Cloud-Dienste«. Dissertation. Technische Universität Dresden, Juli 2014. URL: http://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa-149734 (siehe Seite 48).

[Tie+11] Vincent Tietz, Stefan Pietschmann, Gregor Blichmann u. a. »Towards Task-Based Development of Enterprise Mashups«. In: *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services (iiWAS '11)*. ACM, Dezember 2011, Seiten 325–328. ISBN: 9781450307840. DOI: 10.1145/2095536.2095594. URL: https://dl.acm.org/citation.cfm?id=2095594 (siehe Seiten 2, 9).

[Tie+13] Vincent Tietz, Oliver Mroß, Andreas Rümpel u. a. »A Requirements Model for Composite and Distributed Web Mashups«. In: *Proceedings of the The Eighth International Conference on Internet and Web Applications and Services (ICIW 2013)*. Herausgegeben von Ioannis Moscholios und Marek Rychly. Xpert Publishing Services, Juni 2013, Seiten 75–82. ISBN: 978-1-61208-280-6. URL: http://www.thinkmind.org/index.php?view=article&articleid=iciw_2013_4_10_20161 (siehe Seite 32).

[Tie15] Vincent Tietz. »Aufgabenbasierte Komposition von User-Interface-Mashups«. Dissertation. Technische Universität Dresden, Mai 2015. URL: https://www.amazon.de/dp/3959080522 (siehe Seiten 32, 102, 104, 151).

[Tom+08] Ioan Toma, Douglas Foxvog, Flavio De Paoli u. a. *Non-functional properties in Web services (D28.4 V0.2)*. Herausgegeben von Ioan Toma. WSMO Working Draft. April 2008. URL: http://www.wsmo.org/TR/d28/d28.4/v0.2/ (siehe Seite 47).

[Tre+10] Martin Treiber, Kyriakos Kritikos, Daniel Schall u. a. »Modeling context-aware and socially-enriched mashups«. In: *Proceedings of the 3rd and 4th International Workshop on Web APIs and Services Mashups (Mashups '09/'10)*. ACM Press, Oktober 2010, Seiten 1–8. ISBN: 9781450304184. DOI: 10.1145/1944999.1945001. URL: https://dl.acm.org/citation.cfm?id=1945001 (siehe Seite 40).

[TT08] Vuong Xuan Tran und Hidekazu Tsuji. »QoS Based Ranking for Web Services: Fuzzy Approaches«. In: *2008 4th International Conference on Next Generation Web Services Practices*. Herausgegeben von Ajith Abraham und Sang Yong Han. IEEE Computer Society, Oktober 2008, Seiten 77–82. ISBN: 9780769534558. DOI: 10.1109/NWeSP.2008.41. URL: https://ieeexplore.ieee.org/document/4700385/ (siehe Seite 32).

[Voi15] Martin Voigt. »Kontextsensitive Informationsvisualisierung mit kompositen Rich Internet Applications für Endnutzer«. Dissertation. Technische Universität Dresden, Mai 2015. URL: http://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa-175889 (siehe Seite 32).

[Wat07] Stephen Watt. *Mashups – The evolution of the SOA, Part 2: Situational applications and the mashup ecosystem*. November 2007. URL: https://www.ibm.com/developerworks/webservices/library/ws-soa-mashups2/ (siehe Seite 21).

[Widg12] World Wide Web Consortium. *Packaged Web Apps (Widgets) – Packaging and XML Configuration (Second Edition)*. Herausgegeben von Marcos Cáceres. W3C Recommendation. W3C, November 2012. URL: https://www.w3.org/TR/widgets/ (siehe Seiten 14, 28).

[WSCDL05] World Wide Web Consortium. *Web Services Choreography Description Language Version 1.0*. W3C Candidate Recommendation. W3C, November 2005. URL: https://www.w3.org/TR/ws-cdl-10/ (siehe Seite 47).

[WSMO05] World Wide Web Consortium. *Web Service Modeling Ontology (WSMO)*. W3C Member Submission. W3C, Juni 2005. URL: https://www.w3.org/Submission/WSMO/ (siehe Seiten 37, 47).

[WSPol07] World Wide Web Consortium. *Web Services Policy 1.5 – Framework*. W3C Recommendation. W3C, September 2007. URL: https://www.w3.org/TR/ws-policy/ (siehe Seiten 29, 37, 47).

[Yag88] Ronald R. Yager. »On Ordered Weighted Averaging Aggregation Operators in Multicriteria Decisionmaking«. In: *IEEE Transactions on Systems, Man, and Cybernetics* 18.1 (Januar 1988), Seiten 183–190. ISSN: 0018-9472. DOI: 10.1109/21.87068. URL: https://ieeexplore.ieee.org/document/87068/ (siehe Seite 71).

[Yah11] Yahoo! Inc. *About Pipes*. 2011. URL: https://web.archive.org/web/20150101172637/http://pipes.yahoo.com/pipes/ (siehe Seiten 1, 16).

[Zad65] Lotfi A. Zadeh. »Fuzzy Sets«. In: *Information and Control* 8.3 (1965), Seiten 338–353. ISSN: 0019-9958. DOI: 10.1016/S0019-9958(65)90241-X. URL: http://www.sciencedirect.com/science/article/pii/S001999586590241X (siehe Seite 63).

Stichwortverzeichnis

A
Aktionen, 110
Anforderungen
    an die Aggregation, 69
    an die Modellierung, 56
    aus den Szenarien, 18
Anforderungsassistent, 91, 130
Anwendungen
    Implementierung, 124

B
Browser
    für Anwendungen, 134
    für Komponenten, 135

C
Concerns, 59
CRUISE, 15, 31, 42, 125

D
Definition
    Qualitätsbegriffe, 25

E
Empfehlungssystem, 41, 42
Entwicklungsprozess, 102
Evaluator
    Implementierung, 126

F
Fuzzy-Mengen
    Editor, 132
    Grundlagen, 63
    Implementierung, 126
    Konfiguration, 64
    Werkzeuge, 32

K
Kontextsensorik, 128
Konverter, 127

L
Live-Sophistication, 13, 17, 18

M
Metamodell
    Überblick, 54
    Qualitätsanforderungen, 83
    Qualitätseigenschaften, 58

N
Nutzerstudie, 135

R
Referenzmodell
    Aktionen, 110
    Concerns, 59
    Ereignisse, 88
    Qualitätseigenschaften, 72
Rollen, 21

S
SMCDL, 65, 123
Szenarien, 18

W
Werkzeuge, 130